UNIVERSITY of CALIFORNIA
SANTA CRUZ

MULTIPATH ROUTING MECHANISMS FOR TRAFFIC ENGINEERING AND QUALITY OF SERVICE IN THE INTERNET

A thesis submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Srinivas Vutukury

March 2001

The thesis of Srinivas Vutukury is approved:

Professor J.J. Garcia-Luna-Aveces, Chair

Professor Phokion Kolaitis

Professor Charlie McDowell

Dean of Graduate Studies

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAR 2001		**00-03-2001 to 00-03-2001**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Multipath Routing Mechanisms for Traffic Engineering and Quality of Service in the Internet	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**152**	

Contents

List of Figures v

List of Tables vii

Abstract viii

Acknowledgements x

Dedication xii

1 Introduction 1
- 1.1 Why Multipaths? . . . 5
- 1.2 Traffic Engineering based Optimal Routing . . . 9
- 1.3 Providing Scalable Guaranteed Services . . . 11
- 1.4 Organization of the Thesis . . . 13

2 Multipath Routing Algorithms 15
- 2.1 Problem Formulation . . . 16
- 2.2 Multipath Distance Vector Algorithm . . . 18
  - 2.2.1 Description of MDVA . . . 21
  - 2.2.2 Correctness Proof of MDVA . . . 24
- 2.3 Multipath Link State Algorithm . . . 30
  - 2.3.1 Description of PDA . . . 30
  - 2.3.2 Correctness Proof of PDA . . . 35
  - 2.3.3 Description of MPDA . . . 39
  - 2.3.4 Correctness Proof of MPDA . . . 42
- 2.4 MPATH Routing Algorithm . . . 44
  - 2.4.1 Description of MPATH . . . 46
  - 2.4.2 Correctness Proof of MPATH . . . 50
  - 2.4.3 Complexity Analysis of MPATH . . . 53
- 2.5 Performance Comparison . . . 54
- 2.6 Related Work . . . 61
  - 2.6.1 Distance-vector algorithms . . . 61
  - 2.6.2 Link-state algorithms . . . 62

3 Traffic Engineering based on Minimum-delay Routing **64**
3.1 Minimum-delay Routing Problem Formulation 65
3.2 A Minimum Delay Routing Algorithm 67
3.3 Near-optimal Routing Framework 70
3.4 Implementation of Near-optimal Routing Framework 72
3.4.1 Distributing Traffic over Multiple Paths 72
3.4.2 Computing Link Costs 75
3.5 Performance of Near-optimal Routing framework 76
3.5.1 Performance under Stationary Traffic 79
3.5.2 Effect of Tuning Parameters T_l and T_s 79
3.5.3 Performance under Dynamic Traffic 81
3.6 Minimizing Out-of-Order Packet Delivery 85
3.6.1 Forwarding Datagram traffic 86
3.6.2 Forwarding TCP traffic 88
3.6.3 Hybrid packet forwarding 90
3.6.4 Performance of TE Scheme 92
3.7 Summary 96

4 Multipaths in Guaranteed Services Architecture **97**
4.1 SMART Overview 100
4.1.1 Flow Aggregation 101
4.1.2 Multipaths 104
4.1.3 Packet Forwarding 104
4.1.4 Signaling 106
4.2 Flow Aggregation: Non-fluid Model 107
4.3 Enhanced Multipaths 111
4.4 End-to-End Delay Bounds 113
4.4.1 Multipath Flows 113
4.4.2 Single-path Flows 114
4.5 Quality of Service Routing 115
4.5.1 Path Selection Schemes 117
4.5.2 Call Blocking Rates 118
4.6 Reservation Maintenance Protocol 121
4.7 Related Work in Resource Management 126

5 Summary and Future Work **130**
5.1 Contributions 130
5.2 Future Work 133

Bibliography **136**

List of Figures

1.1 Example network. Links have unit bandwidth. 7

2.1 Distance vector processing in MDVA. 23
2.2 The Partial-topology Dissemination Algorithm 30
2.3 Neighbor topology table update procedure in PDA 31
2.4 Main topology table update procedure in PDA 32
2.5 Illustration of the main table update procedure of PDA. 34
2.6 Illustrating the significance of the tie-breaking rule. 35
2.7 Multiple-path Partial-topology Dissemination Algorithm (MPDA) 40
2.8 Active-passive phase transitions in MPDA. 42
2.9 The PATH Algorithm 46
2.10 Neighbor table update algorithm in PATH 46
2.11 Main table update procedure of PATH protocol 47
2.12 Multi-path Loop-free Routing Algorithm 48
2.13 Topology used in simulations 55
2.14 Average convergence times. $\alpha = 1$, $\beta = 5$. 56
2.15 Average message overhead. $\alpha = 1$, $\beta = 5$. 56
2.16 Average convergence times. $\alpha = -0.1$, $\beta = -0.4$. Note that DBF and MDVA behave identically. 57
2.17 Average message overhead. $\alpha = -0.1$, $\beta = -0.4$. Note that DBF and MDVA behave identically. 57
2.18 PDF of convergence times. $\alpha = 1$, $\beta = 5$, $k = 1$. 58
2.19 PDF of message overhead. $\alpha = 1$, $\beta = 5$, $k = 1$. 58
2.20 PDF of convergence times. $\alpha = 5$, $\beta = 5$, $k = 1$. 59
2.21 PDF of message overhead. $\alpha = 5$, $\beta = 5$, $k = 1$. 59
2.22 PDF of convergence times. $\alpha = 0$, $\beta = -0.4$. Note that DBF and MDVA behave identically. 60
2.23 PDF of message overhead. $\alpha = 0$, $\beta = -0.4$. Note that DBF and MDVA behave identically. 60

3.1 Heuristic for initial load assignment. 73
3.2 Heuristic for incremental load adjustment. 74
3.3 Topologies used in simulations 77
3.4 Delays of OPT and MP in CAIRN. 78

3.5 Delays of OPT and MP in NET1. 78
3.6 Delays of MP and SP in CAIRN. 80
3.7 Delays of MP and SP in NET1. 80
3.8 Delays when T_s is kept constant and T_l is increased in CAIRN. 82
3.9 Delays when T_s is kept constant and T_l is increased in NET1. 82
3.10 (a) Step response in NET1 using OPT and MP routing. (b) Variable input traffic pattern 83
3.11 84
3.12 Forwarding UDP traffic according to routing parameters. 88
3.13 Forwarding TCP packets according to routing parameters. 91
3.14 Comparison of single path and TE scheme 92
3.15 Comparison of single path and TE scheme 93
3.16 Comparison of single path and TE scheme 94
3.17 Comparison of single path and TE scheme 94
3.18 (a) Delays as a function of volume of flows 95
3.19 Delays in presence of hybrid traffic 96

4.1 (a) A sample network (b) Multipath successor graph 105
4.2 Block diagram for the SMART router 106
4.3 (a) Dynamic Token Bucket (b) Distributor 107
4.4 (a) An uneven multipath (b) Equal cost multipath (c) enhanced multipath . . 113
4.5 Call-blocking rates as a function of ρ. $\beta = 20$ 120
4.6 Call-blocking rates when flows are divided into several smaller flows 121
4.7 Event Handling in the AGREE protocol 124

List of Tables

1.1 Tradeoffs under different schemes 8

2.1 Preferred neighbor table 34

Abstract

Multipath Routing Mechanisms for Traffic Engineering and Quality of Service in the Internet

by

Srinivas Vutukury

The success of the IP architecture is largely due to the simplicity, robustness and scalability that resulted from its the *connection-less* design methodology. As the Internet evolves, it must support new services such as QoS, and when extensions are made to the IP architecture to support such services, its basic connection-less model must be preserved to retain the scalability and robustness that made it so successful. In the past few years, with the Internet becoming the main communication infrastructure, IP networks are faced with two challenging problems that require immediate attention: *traffic engineering* and *supporting guaranteed services*. Providing efficient, robust and scalable solutions to these problems within the framework of the connection-less IP has become extremely important and urgent.

The traffic engineering problem arose mainly because the single-path routing prevalent in IP networks proved very inefficient in the face of rapid growth of the Internet. To improve performance of current IP networks, several solutions based on the multi-protocol label-switching (MPLS) have been proposed by IETF. The main idea in these approaches is to setup alternate paths using label-switching, and distribute traffic over them. There are couple of serious concerns with these approach. First, most of the proposed solutions are not based on any theoretical results on optimal routing. Second, using *connection-oriented* technology like virtual circuits or MPLS violates its connection-less methodology of IP that contributed to the very success of the Internet. These approaches tend to replace the IP architecture,

rather than evolve it. We propose a solution to the traffic engineering problem that addresses these concerns. The key idea in our approach is to use *multipaths* to implement "near-optimal" routing, while keeping the scalability of data and control plane mechanisms similar to that in today's IP routing. More importantly, the proposed approach preserves the connection-less nature of the IP architecture.

Today, there is a growing need to support real-time applications that require delay and bandwidth guarantees. To address this, the IETF proposed the Intserv architecture and the associated RSVP. This architecture does not scale well to backbone networks that carry large numbers of flows because of the *per-flow* reservations and per-flow processing that is required in the routers. Several other architectures have also been proposed to support guaranteed services, but most of them are either inefficient in terms of bandwidth utilization, or use *connection-oriented* approaches such as virtual circuits and MPLS. The main concern with all these architecture, besides scalability, is that they introduce connection-oriented mechanisms in the IP architecture, thus compromising its robustness. We base our approach to this problem on multipaths, and provide a complete solution through a novel architecture, SMART, which adheres to *connection-less* nature of the IP architecture. We show how multipaths can help achieve scalability and performance of the network, without sacrificing the kind of service guarantees that can be offered to end users.

Acknowledgements

I thank my advisor J.J. Garcia-Luna-Aceves for all the guidance and support he has given me during the last four years. I am grateful to him for believing in my abilities and giving me full freedom to explore the field while providing invaluable advice on the way. His knowledge, energy and enthusiasm have been a constant source of inspiration for me.

I thank members of my committee Charlie McDowell, Phokion Kolaitis and William Zaumen for their advice and feedback. Charlie's inquisitive questions have made be think and present the ideas better. Phokion Kolaitis is one of the best teachers I have had, and his theory classes have taught me how to construct rigorous proofs, which served me well in my research. I also wish to thank Anujan Varma, Richard Hugley, Allen Van Gelder, Darrell Long, Craig Wittenbrink and Ira Pohl, from whom I had the opportunity to learn during my graduate study. I would also like to thank Carol Mullane in the Graduate Office for her friendly conversations, and for her timely reminders of imminent deadlines.

I thank my fellow CCRG group members Marcelo, Makis, Jyoti, Chris, Mike, Ewerton, Soumya, Lichun, Lori, and Brad for providing an intellectually stimulating environment in the group, and for being good friends.

My wife Padmaja, to whom this thesis is dedicated, was instrumental in my returning to graduate school. Life rarely offers a second chance, and I thank her for providing me this opportunity to pursue graduate study. I am grateful for the constant support and encouragement she had given me during these years.

I thank my mother-in-law for helping us enormously in taking care of our baby daughter Sruti during the past two years. I thank my parents for their well-wishes and blessings. I thank my family members Sathi, Swarupa and Vinod for always helping me out. I thank my friends Ramu, Chandra, Venkat, Prasad, Venu, Vani, Malini, Shanti, Vijaya and Roja for

their warm friendship. Finally, I thank my little daughter Sruti and her friends, Teju, Manasa, Apurva, Kavinanda, Pranit and Pranathi for giving me so much joy when the going was tough.

I would not have progressed an inch on this journey without the blessings of Lord Sri Sai Baba. From his lotus feet flowed all the ideas in this thesis; I only received them faithfully.

This work was supported in part by the Defence Advanced Research Projects Agency (DARPA) under grants N6601-00-1-8942, F30602-97-2-0338 and F19628-96-C-0038.

For Padmaja.

Chapter 1

Introduction

The Internet Protocol (IP), true to its name, has become the predominant protocol of the Internet. The tremendous success of IP networks is largely due to the simplicity, flexibility, scalability and robustness of its network layer design. In an IP network, a router maintains a simple table specifying the next-hop router for each destination and uses a simple first-come first-serve discipline for forwarding packets at the links. By delegating most of the complex functionality, such as reliable and in-order packet delivery (e.g., TCP), to higher layers of the protocol stack, the network layer itself is kept simple and flexible. IP networks are very scalable as the routers maintain minimal state in the routers and require minimal per-packet processing. IP networks are very robust, because IP routers quickly recompute paths to destinations when links and nodes fail in a way that is completely transparent to the higher layers. These benefits are mainly due to the *connection-less* methodology that is adopted in the IP architecture [12]. An IP packet contain information that is globally interpretable and sufficient to forward it based on the most current information in the routers determined by *distributed routing protocols*. This is in sharp contrast with connection-oriented technologies, such as ATM and MPLS, which use

labels embedded in the packets that have only local interpretation in the network and the use state in the routers that is signaled from the edge. For this reason, connection-oriented approaches like ATM and MPLS tend to lack scalability and robustness of the connection-less IP approach. As the Internet evolves, enhancing the IP architecture with extensions that support new services become inevitable, and when such extensions are made, caution must be exercised to preserve the basic connection-less paradigm of IP technology. Following this design principle, we address two important problems that have received tremendous attention in recent times by the Internet community: *Traffic Engineering* and *Quality of Service*. Many solutions have already been proposed, but most of them introduce "virtual-circuits" through use of *labels* or *dynamic state* in the packets that have only local interpretation. While such methods offer short-term solutions, unfortunately, their long-term viability is highly suspect. Our thesis is that *using multipaths, robust and scalable solutions to traffic engineering and guaranteed services can be provided within the connection-less model of IP, without resorting to connection-oriented mechanisms such as virtual-circuits and MPLS.*

We observe that solutions based on label-switching or virtual-circuits have proliferated mainly because there is an urgent need to improve network performance by using *alternate paths* for packet forwarding, and while establishing paths other shortest paths in current IP routing seemed relatively difficult, the same is quite natural in connection-oriented architectures. So, if alternate forwarding paths between source destination pairs can somehow be established and maintained with ease, solutions to several of the problems can be provided within the connection-less routing framework. This is where *multipaths* prove their potential. Multipaths are *directed acyclic graphs* with the destination as the sink node, and are a generalization of single shortest paths. Multipaths utilize full connectivity of the network for forwarding packets, and can be established easily using distributed routing protocols unlike connections that must

be signaled from the edge. With minimal changes to the IP routing, we show that multipaths offers remarkable performance without compromising on the simplicity, flexibility, scalability and robustness of IP.

Most routing protocols in today's IP networks construct a single path between each source-destination pair (e.g. RIP, Cisco's EIGRP, OSPF), and single-path routing cannot handle congestion properly and makes poor use of network resources. With the rapid growth of the Internet in the past few years, there is an urgent need to make more efficient use of network resources, such as bandwidth. This challenging problem, referred to as Traffic Engineering, is currently being addressed by the Internet community, and several solutions have already been proposed based on the IETF's traffic engineering framework [2]. In the proposed solutions so far, the key idea is to set up alternate paths between a source and a destination and use them to handle congestion and as detours around network failures. While any method can be used for establishing alternate paths, from a practical point of view, the solutions that are scalable are of importance. Unfortunately, most current approaches propose using MPLS for establishing alternate paths, which are difficult to setup and manage in the current connection-less IP networks. In contrast, we propose techniques using multipaths constructed by distributed routing protocols similar to those in current use, instead of using signaling protocols to establish label-switched paths in the IP cloud. By allocating traffic over multipaths, that are free of loops at every instant, optimal use of network resources can be achieved. We show that our proposed multipath-based network layer routing and packet forwarding mechanisms, not only improve throughput but also lower delays to levels comparable to theoretically optimal routing.

Today, there is a growing need in the Internet to support real-time applications such streaming video, that require delay, loss, bandwidth and delay-jitter guarantees. For this, the

Internet community proposed the Intserv [10, 76] for providing *deterministic* service guarantees in the Internet. The Intserv architecture provides deterministic guarantees using *per-flow* reservations, limiting its scalability. In this dissertation, we address the support of guaranteed services, so we focus mainly on Intserv and other guaranteed services architectures proposed in literature [28, 41, 61]. The scalability problem in Intserv arises mainly because, networks manage reservations of each session individually, and use per-flow fair scheduling at the links. A key design principle in Intserv is to avoid using any virtual circuits and proceed with a connection-less approach of current IP. As a result flows are setup along shortest-paths computed by the underlying routing protocols, and reservations of the flows are managed using the soft-state protocol, RSVP [76]. Unfortunately, these per-flow mechanisms do not scale to backbone networks where there are hundreds of thousands of flows. To address this scalability problem, flows must be aggregated and handled collectively so that the state size and processing power required in the routers is reduced. There have been efforts in this direction. For example, in the aggregated version of RSVP [20], per-flow routing state is replaced with per source-destination routing state, resulting in state size that grows quadratically in number of nodes. This is a significant improvement from per-flow approach. However, while scalability is one problem, performance is another problem. If flows are established only along the shortest paths to enable flow aggregation, bandwidth will soon be consumed on those paths, leading to high call-blocking rates. To improve performance over shortest paths, alternate paths must be used. But using alternate paths setup through virtual-circuits or label-switching technologies will compromise the connection-less approach of IP. In this situation, multipaths hold the key to performance. We show that by aggregating flows along multipaths using flow classes, we can continue to maintain the scalability and connection-less feature of the current IP architecture, while providing enough connectivity to improve performance. In the resulting architecture,

SMART, the size of the state in the routers and the complexity of mechanisms used to provide the guaranteed service depends only on the network parameters and not on the number of end-user flows. Lastly and most importantly, the multipaths are constructed using dynamic distributed routing protocols, and are not signaled from the edge as in connection-oriented architecture.

In summary, providing deterministic service guarantees and improving performance through traffic engineering are two of the most important challenges facing IP networks today, and hence, giving scalable and efficient solutions to these problems is the main focus of this thesis. We show that solutions based on multipaths offer a remarkable tradeoff between scalability and performance. In best-effort architectures, multipaths optimize throughput and minimize end-to-end delays, and in integrated service architectures they improve call-blocking rates. Finally and more importantly, the proposed approaches based on multipath routing conform with the connection-less paradigm of the Internet. The multipaths are constructed using distributed routing algorithms, unlike virtual circuits that are signaled from the edge of the network. In the next section, we describe multipaths and routing parameters in more detail, and in the subsequent sections introduce the key ideas in our solutions.

1.1 Why Multipaths?

In this section we will show using a simple example why multipaths hold a promise compared to virtual circuits from a performance and implementation point of view. Consider the network shown in Fig. 1.1(a). For this network, Fig. 1.1(b) shows the single shortest paths from all nodes to destination j. The corresponding available bandwidths along these paths are shown in column one of Table 1.1. Observe that many of the links $((m,n),(i,n),(i,m))$ in Fig. 1.1(b) are not utilized for delivering packets for destination j. By utilizing the full connectivity

of the network we can see that potentially greater bandwidth can be made available for each source to send traffic to j. This extra bandwidth can be tapped using one of two techniques: virtual-circuits or multipaths. First consider the virtual-circuits in Figs. 1.1(c)-(e) showing the path that are available from i, m and n to destination j. The corresponding available bandwidth for each source is shown in column two of Table 1.1. Even though all the potentially available bandwidth is not available for the source in the presence of traffic between other source-destination pairs, given that traffic load fluctuates with time, each source can gain through *statistical multiplexing* from this extra available bandwidth. Congestion occurs when offered load is greater than the available bandwidth, and because of larger available bandwidth, now the onset of congestion is pushed to a higher offered load. Similarly, in the context of the Integrated Services architecture, the increased available bandwidth between each source-destination pair results in higher call-acceptance rates as compared to the call-acceptance rates when only single-paths are used.

The problem with establishing alternate paths as described above is that these routes have to be "pinned" using virtual circuits or MPLS and sources need to know labels that need to be used to forward packets along these paths. This increases the routing state needed to establish them as shown in Table 1.1 for the example. When virtual-circuits or label-switched paths are used, the number of paths that can potentially pass through a router grow exponentially. Also, virtual-circuits are difficult to construct and maintain as the network topology and traffic conditions change, because they must be signaled from the edge rather than constructed using routing protocols. Also, using virtual circuits or MPLS is a significant departure from the connection-less methodology of IP architecture. Now the question is: "How can we gain in available end-to-end bandwidth while keeping the routing state size and forwarding mechanism similar to the one in current IP architectures?" The answer is multipaths.

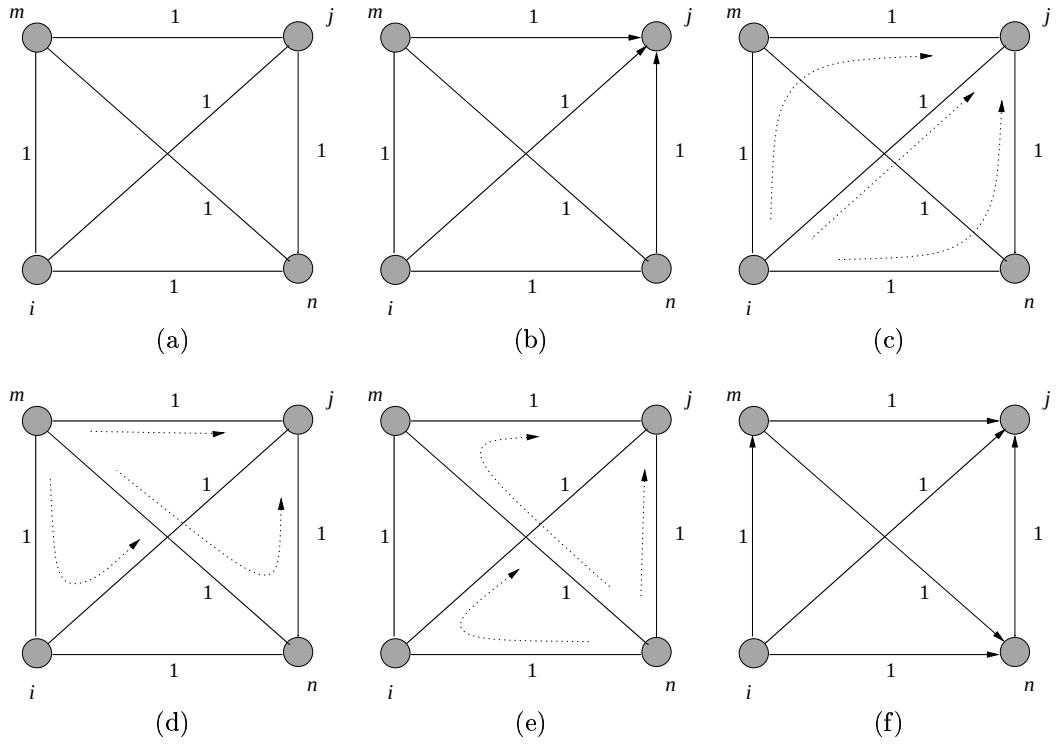


Figure 1.1: Example network. Links have unit bandwidth.

Consider the *directed acyclic graph* or *multipath* for j as shown in Fig.1.1(f), which is basically a generalization of the routing tree of Figure 1.1. The end-to-end available bandwidth from each source for destination j is shown in column three of Table 1.1. Observe that the bandwidths are greater than that of single-path routing, but are lower than those of virtual circuits of Fig. 1.1(c)-(e). But, what is important is that the multipath scheme is scalable and easy to implement compared to virtual-circuits. The advantage of the multipaths is that they are constructed using distributed routing protocols and can be easily implemented as an extension of single-path routing. The single next-hop is extended to a set of next-hop neighbors and with each next-hop neighbor a *routing parameter* is associated that specifies the amount of traffic of the given destination that must be forwarded to the corresponding neighbor. The

Available Bandwidth			
Source-destination pair	Single paths	Virtual Circuits	Multipaths
(i, j)	1	3	3
(m, j)	1	3	2
(n, j)	1	3	1
State Size			
Node	Single paths	Virtual Circuits	Multipaths
i	1	5	1
m	1	5	1
n	1	5	1

Table 1.1: Tradeoffs under different schemes

routing table entry now looks as follows: $\langle dst, \{nbr_1, .., nbr_k\}, \{\phi_1, .., \phi_k\}\rangle$. The amount of traffic that neighbor nbr_i is ϕ_i and $\sum \phi_i = 1$. The number of entries in a routing table remain the same. Only the size of entry is increased, but is bounded by the number of neighbors.

Consider a network with N nodes (routers) and L edges (links), and let N^i be the set of neighbors of node i. On the control plane, the problem consists of finding at each router i for each destination j, the next-hop or the successor set $S_j^i \subseteq N^i$ and the corresponding routing parameter set $\Phi_j^i = \{\phi_{jk}^i | k \in S_j^i\}$. On the data plane, the packet forwarding engine is extended from single next-hop look-up based on destination to allocating traffic in proportion to routing parameters using a weighted round robin discipline. The complexity of packet forwarding decision is increased, but depends only on the number of neighbors of the router. When router i receives a packet for destination j, it forwards the packet to one of the neighbor routers in the successor set S_j^i according to Φ_j^i. If the routing graph SG_j, a directed subgraph of G, is defined by the link set $\{(m, n) | n \in S_j^m,\ m \in N\}$, a packet destined for j follows a path in SG_j. The goal is to construct SG_j and the routing parameters Φ_j^i such the traffic is load-balanced in an optimal way. The methods and metrics used to compute and maintain the multipaths depends on the problem. In the traffic engineering problem, the multipaths and routing parameters

reflect those in optimal routing. In the scalable QoS routing architecture, the multipaths are constructed based on hop-counts and the routing parameters are set according to reservations of flows. The next two sections describe the problems in more detail.

1.2 Traffic Engineering based Optimal Routing

Most routing protocols in current use, such as RIP [35], EIGRP [1] and OSPF [45], make very inefficient use of bandwidth usage. To improve bandwidth utilization, and reduce delays, several improvements to the basic routing protocols have been proposed [16, 45, 64, 66]. However, these improvements have been largely ad hoc and lack any theoretical foundation. For example, in ECMP [45] load is distributed equally over multiple equal-cost paths typically using simple round-robin distribution. In OSPF-OMP [66] traffic distribution on multiple paths is based on heuristics. To make optimal use of network resources and minimize delays, traffic between source-destination pairs have to be allocated along multiple paths in proportions that reflect optimal routing [5]. Practical implementations of optimal routing, such as Codex [36], use virtual circuits to setup up flows that reflect an optimal distribution, but the architecture introduces unacceptable complexity in the routers and is not scalable. Practical implementations of optimal routing that are scalable is still elusive.

Recently, Traffic Engineering (TE), has received tremendous attention in the Internet community and several IETF drafts have appeared [2, 42, 62]. Typically in a TE approach, the goal is to determine a set of flows with associated paths and bandwidths that meet the optimization criteria for a given input traffic matrix. TE solutions are most effective in a network under a single administrative domain such as ISPs, where knowledge of the link characteristics and input traffic matrix can be obtained. Traffic engineering has a wide scope and covers diverse forms of routing that address survivability, QoS and policy-based routing. However,

congestion management and bandwidth utilization are of pressing importance and generally more difficult to address, and thus is the problem we focus in this paper. Despite the initial failure of direct introduction of minimum-delay routing in networks, a traffic engineering approach to minimum-delay routing seems to have great potential for success. To the best of our knowledge, this is the first attempt to construct a TE system based on minimum-delay routing.

There are two parts to the proposed solution: (1) obtain input traffic matrix and the link characteristics, and determine multipaths and routing parameters that optimize end-to-end delays, and (2) on the data plane, forward packets according to routing parameters. There are several algorithms available to solve the minimum-delay routing problem [4, 57]. Any one of them can be used to obtained the solution in the form of routing parameters, and then downloaded into the routers using a routing protocol or through a signaling protocol. On the other hand, the routing parameters can be obtained using on-line measurements [68]. Using off-line computation can be more accurate, but lack the adaptiveness of on-line measurements, and moreover may require some signaling in some cases. While either way can be used to realize practical implementation of optimal routing, our preferred approach is to employ the on-line approach. In both cases, our TE approach to optimal routing scales much better compared to other methods based on MPLS. Setting up explicit paths from the source to the destination using label-switched paths as in MPLS-OPT [65] leads to complexity at the edge nodes. On the other hand, if splitting of flows is not allowed, to simplify the MPLS implementation, the solutions obtained are sub-optimal [72]. Moreover, using connection-oriented approach like MPLS contradicts the connection-less paradigm of the Internet. However, not only optimal routing, but even traffic engineering problems based on other optimization objectives can be expressed in terms of multipaths and routing parameters.

To obtain an accurate and scalable implementation of an optimal routing on the data plane, we combine ideas from Differential Services model [25] and OSPF-OMP [66] with new mechanisms. Once the routing parameters are obtained, achieving a fine granularity in distribution of traffic on the data plane is a challenge. In the proposed technique, a randomly generated key that is unique to a connection is generated and inserted in each packet at the ingress node. Packets are forwarded in the intermediate routers based on the key and the routing parameters. This method is better than the hashing approach of [66] as it distinguishes between many connections that may exist between a source-destination pair. Also, datagram packets are treated differently than traffic that requires in-order delivery. By using datagram packets to be forwarded along paths that less loaded according to routing parameters more accurate distribution of actual traffic is achieved and, consequently, the end-to-end delays are closer to the optimal. The architecture is practical and can be implemented in conjunction with other per-hop behaviors of the Diffserv architecture [70]. Our proposed TE solution, therefore, achieves two objectives: it approximates the minimum-delay routing, and it uses the connection-less methodology of IP.

1.3 Providing Scalable Guaranteed Services

Designing a scalable and efficient architecture to support guaranteed services is a challenging problem that is currently being addressed by the Internet community. The Intserv architecture, proposed by IETF, uses *per-flow* bandwidth reservations and uses per-flow state in the routers, which is not scalable to backbone networks where there are large number of flows. To realize a scalable guarantees service architecture in IP all of the following mechanisms must be scalable: (1) link scheduling, (2) state size in the routers, (3) QoS path selection, and (4) reservation management protocol. At the same time, the architecture should be efficient

in bandwidth utilization, so that the call-acceptance rate is high. Moreover, the architecture should be based on a connection-less model to maintain the flexibility and robustness of the Internet. With these objectives, we a complete solution to the problem through a novel architecture we call SMART. The key idea is to *aggregate flows along multipaths.* The result is a scalable architecture that is not only efficient and scalable, but also conforms with the connection-less paradigm of the Internet.

The challenge to flow aggregation is that it should be performed such that deterministic guarantees individual flows in spite of lacking complete isolation between flows. For aggregating flows, we introduce the concept of *burst-ratio* and use it to classify flows into a small number of flow classes. Within the core, routers handle traffic only on the basis of class. The fair link schedulers (WFQ, for example) arbitrate packets from a small number of queues corresponding to the flow classes. By performing only per-class processing within the network, assurances for each individual flow can be met. The burst-ratio technique is a powerful aggregation technique that aggregates large numbers of flows into few classes. As a result, the size of reservation state in the routers in the SMART architecture is a linear function of number of destinations.

All approaches to date use a single path from source to destination for making reservations for a flow. This results in high call-blocking rates. The key to improving call-acceptance rates without resorting to alternate paths setup via MPLS, is to use multipaths. We show that using multipaths, the size of the routing tables can be made to grow linearly with the number of destinations, without compromising much on call-acceptance rates. A major consequence of this is that the soft-state based reservation maintenance protocol, AGREE, is highly scalable as the it uses only per-class per-destination refresh messages.

Determination of QoS capable routing paths is a complex task, and all approaches to

date require flooding of information regarding available bandwidth on the links of the network, which is not only expensive to begin with but often the information is outdated at the time the QoS paths are determined. The proposed path selection scheme in SMART architecture does not require flooding of link utilization information in the network and yet, it provides comparable performance to benchmark schemes in terms of call-acceptance. Another significant advantage of the proposed aggregation scheme is that it mitigates bandwidth fragmentation in the network. When bandwidth sizes are large, the single path approach results in bandwidth fragmentation. By dividing large bandwidth flows into smaller bandwidth flows, greater bandwidth utilization can be achieved in the SMART architecture without additional state in the routers. Overall, the SMART architecture demonstrates that multipaths combined with routing parameters and flow classes is a powerful new approach to support guaranteed services in IP networks.

1.4 Organization of the Thesis

This thesis is divided into three parts: multipath routing protocols, traffic engineering and scalable guaranteed services architectures. The three parts of the thesis are organized into three chapters. Here is the chapter by chapter outline of the thesis.

In Chapter 2, three multipath routing protocols: MPDA, MDVA and MPATH are presented using a unified approach to instantaneous loop-freedom based on a set of Loop-free invariants (LFI). The protocols differ in the type of control messages the routers use to communicate; MPDA is a link-state routing protocol, MDVA is purely a distance-vector protocol, and MPATH uses predecessor information in addition to distance information. Each protocol is described and proved formally. Later in the chapter, the three protocols are compared with each other, and with protocols RIP and OSPF, using message overhead and convergence times

as performance metric.

In Chapter 3, we present the first application of a multipath protocol in the context of traffic engineering, and show how near-optimal delays can be achieved. We begin the chapter with the description of the minimum-delay routing protocol (MDRP) and problems inherent in its implementation. We derive an approximation to MDRP that fixes this drawback, yet offers near-optimal delays. We then show how this framework can be realized in practice using the multipath routing protocols described in Chapter 2, and combine with traffic load-balancing that mimics the traffic load-balancing of MDRP. In the final section of the chapter we present results of simulations that show how the framework provides end-to-end delays that are comparable to MDRP.

In Chapter 4, we focus on guaranteed services architectures. We start by identifying the main drawback of the current Integrated Services architecture, namely scalability and performance. We first show how the routing and reservation state can be reduced using the multipaths and flow aggregation classes in the context of the fluid flow model. We then extend the concepts to a non-fluid model and derive the end-to-end delay bounds. The key to scalability is that all control plane and data plane mechanisms must have complexity that is no more than a linear function of the network parameters. Accordingly, we present the reservation maintenance protocol AGREE that has refresh message overhead that scales linearly with the number of destinations. In the last section, we describe a simple scalable path selection protocol that performs as well as a benchmark path selection scheme, such as the WSP, without using any link advertisement.

Finally in Chapter 5, we summarize our contributions and suggest interesting research directions.

Chapter 2

Multipath Routing Algorithms

This chapter presents three routing algorithms for constructing multipaths: (1) *Multipath Distance Vector Algorithm* (MDVA), a distance vector algorithm, (2) *Multipath Partial Dissemination Algorithm* (MPDA), a link-state algorithm, and (3) MPATH, a distance vector algorithm combined with predecessor information. A key feature of these routing algorithms is that they provide instantaneous loop-freedom, which, as we will show in the next chapter, is critical to implementing traffic engineering solutions based on minimum-delay routing. Also, the instantaneous loop-freedom in the three algorithms is treated in a uniform manner through a set loop-free invariants.

We start with formal specification of the loop-free multipath routing problem and state the safety and liveness conditions that the routing algorithms must satisfy. We then describe each algorithm in detail and provide their correctness proofs. Finally, we compare the performance of these algorithms with the widely used routing protocols RIP and OSPF, and end the chapter with a brief survey of related routing algorithms.

2.1 Problem Formulation

Let a computer network be represented as a graph $G = (N, L)$, where N is set of nodes (routers) and L is the set of edges (links), and let N^i be the set of neighbors of node i. The problem consists of finding the successor set at each router i for each destination j, denoted by $S_j^i \subseteq N^i$, so that when router i receives a packet for destination j, it can forward the packet to one of the neighbor routers in the successor set S_j^i. By repeating this process at every router, the packet is expected to reach the destination. If the routing graph SG_j, a directed subgraph of G, is defined by the link set $\{(m,n)|n \in S_j^m,\ m \in N\}$, a packet destined for j follows a path in SG_j. Two criteria determine the efficiency of the routing graph constructed by the protocol: *loop-freedom* and *connectivity*. It is required that SG_j be free of loops, at least when the network is stable, because routing loops degrade network performance. In a dynamic environment, a stricter requirement is that SG_j be loop-free at *every instant*, i.e., if S_j^i and SG_j are parameterized by time t, then $SG_j(t)$ should be free of loops at any time t. This is the *safety property* that is enforced in the proposed multipath routing protocols. If there is at most one element in each S_j^i, then SG_j is a tree and there is only one path from any node to j. On the other hand, if S_j^i's have more than one element, then SG_j is a directed acyclic graph (DAG) and has greater connectivity than a simple tree, enabling traffic load balancing.

Given that there are potentially many directed acyclic graphs (DAGs) for a given destination in a graph, a question arises as to which DAG should be used as a routing graph? Firstly the routing graph should be uniquely defined and secondly it should be easily computable by a distributed algorithm. The natural choice for the routing graph is the one defined by the shortest paths. Accordingly, we defines $S_j^i(t) = \{k|D_j^k(t) < D_j^i(t), k \in N^i\}$, where D_j^i is the cost of the shortest path from i to j measured as the sum of the costs of the links on the path. The routing graph SG_j implied by this set is unique and is called the *multipath*. To

compute D_j^i, distributed routing algorithms may exchange any information (distance-vectors or link-states). But they must ensure that the D_j^i's *converge* to the correct distances. Convergence is formally defined as follows. Let $\mathbf{G}(t)$ denote the topology of the network as seen by an "omniscient observer" at time t, and $\mathbf{D}_j^i(t)$ denote the distance of i to j in $\mathbf{G}(t)$. We use bold font to refer to all quantities in $\mathbf{G}$. Assume the network has stable configuration up to time t. We say the network has converged to the correct values at t if $D_j^i(t) = \mathbf{D}_j^i(t)$ for all i and j. Now, if a sequence of link cost changes occur between t and t_c and none after t_c, then the routing algorithm is said to converge if at some time $t_c \leq t_f < \infty$, $D_j^i(t_f) = \mathbf{D}_j^i(t_f) = \mathbf{D}_j^i(t_c)$. We call this the *liveness property*. In addition, during the convergence phase, the algorithm must ensure the safety condition that $SG_j(t)$'s are loop-free at every instant t.

How can we ensure that SG_j is always loop-free? To do this we use a new variable FD_j^i, called the feasible distance, which is an 'estimate' of the distance D_j^i in the sense that FD_j^i is equal to D_j^i when the network is in stable state, but to prevent loops during periods of network transitions, it is allowed to differ temporarily from D_j^i. Let D_{jk}^i be the distance from k to j as reported to i by k. To ensure loop-freedom at every instant, FD_j^i, D_{jk}^i and S_j^i must satisfy the Loop-Free Invariant (LFI) conditions. The LFI conditions capture all previous loop-free conditions [30, 74] in a unified form that simplifies protocol design and correctness proofs.

Loop-free Invariant Conditions(LFI):

$$FD_j^i(t) \leq D_{ji}^k(t) \qquad k \in N^i \tag{2.1}$$

$$S_j^i(t) = \{\, k \mid D_{jk}^i(t) < FD_j^i(t) \} \tag{2.2}$$

The invariant conditions (2.1) and (2.2) state that, for each destination j, a node i can choose a successor whose distance to j, as known to i, is less than the distance of node i to j that is known to its neighbors.

Theorem 1 *If the LFI conditions are satisfied at any time t, the $SG_j(t)$ implied by the successor sets $S_j^i(t)$ is loop-free.*

Proof: Let $k \in S_j^i(t)$ then from (2.2) we have

$$D_{jk}^i(t) \quad < \quad FD_j^i(t) \tag{2.3}$$

At node k, because node i is a neighbor, from (2.1) we have $FD_j^k(t) \leq D_{jk}^i(t)$. Combining it with (2.3) we get

$$FD_j^k(t) \quad < \quad FD_j^i(t) \tag{2.4}$$

Eq.(2.4) states that, if k is a successor of node i in a path to destination j, then k's feasible distance to j is strictly less than the feasible distance of node i to j. Now, if the successor sets define a loop at time t with respect to j, then for some node p on the loop, we arrive at the absurd relation $FD_j^p(t) < FD_j^p(t)$. Therefore, the LFI conditions are sufficient for loop-freedom. □

The above theorem suggests that any distributed routing protocol (link-state or distance-vector) attempting to find loop-free shortest multipaths must compute D_j^i, FD_j^i and S_j^i such that the LFI conditions are satisfied, and such that at convergence $D_j^i = FD_j^i =$ minimum distance from i to j. Based on these conditions we design the three multipath routing algorithms.

2.2 Multipath Distance Vector Algorithm

According to the Distributed Bellman-Ford (DBF) algorithm, each node i repeatedly executes the equation $D_j^i = min\{D_{jk}^i + l_k^i \mid k \in N^i\}$ for a destination j, and each time D_j^i

changes it reports it to all its neighbors. A known property of DBF is that it always converges, and converges fast, when link costs only decrease [38]. However, convergence is not assured if link-costs increase. Furthermore, when link failures result in network partitions, DBF never converges. This is the well-known *counting-to-infinity problem* [63]. Intuitively, the count-to-infinity problem results due to "circular" computation of distances; that is, a node computes its distance to destination using a distance communicated by a neighbor, which happens to be the length of the path that runs through the node itself. The node using such a distance is unaware of this because nodes only exchange distance information and no path information.

Circular computation of distances that occur in DBF can be prevented if distance information is propagated along a DAG rooted at a destination. Given a DAG, each node computes its distance using distances reported by the "downstream" nodes and reports its distance to "upstream" nodes. This method called *diffusing computations* was first suggested by Dijkstra et al [27] to ensure termination of distributed computation; a diffusion computation always terminates due to the acyclic ordering of the nodes. DUAL [30], the algorithm on which EIGRP [1] is based, uses diffusing computation to solve the count-to-infinity problem. In addition to DUAL, several distance vector algorithms have been proposed that use diffusing computation to overcome the counting-to-infinity problem of DBF [56, 44, 38, 74]. The algorithm suggested by Jaffe and Moss [38] allows nodes to participate in multiple diffusing computation of the same destination and requires use of unbounded counters, for which reason it may not be practical. In contrast, a node in DUAL and DASM [74] participates in only one diffusing computation for any destination at any one time and thus requires only a toggle bit. MDVA presented here follows the second approach.

Two questions arise regarding diffusing computation:

1. Because there are potentially many DAGs for a given destination, which one should be

used for diffusing computation?

2. How should diffusing computation be carried out in a dynamic environment in which the chosen DAG changes with time?

The answer to the first question is straightforward: the shortest multipath SG_j is the right choice given that computing SG_j is our final goal. The second question is not as straightforward. A SG_j used for carrying out a diffusing computation can be allowed to change if the following conditions hold: (1) SG_j is acyclic at *every instant*, and (2) at any instant, if a node reports a distance through a neighbor k in S_j^i it must ensure that k remains in S_j^i until the end of the diffusing computation. That these conditions prevent circular computation of distances follows from this argument. Assume that a circular computation occurs at time t involving nodes i_0, i_1, .. i_m. Let a node i_p, $1 \leq p \leq m$, compute its distance at $t_p \leq t$ using distance reported by i_{p-1}, and i_0 computes its distance using the distance reported by i_m at t_0. Because i_{p-1} is held in the successor set of i_p for $1 \leq p \leq m$ and i_0 holds i_m until the diffusing computation ends, we have

$$
\begin{aligned}
i_0 \in S_j^{i_1}(t_1) &\implies i_0 \in S_j^{i_1}(t) \\
i_1 \in S_j^{i_2}(t_2) &\implies i_1 \in S_j^{i_2}(t) \\
&\vdots \\
i_{m-1} \in S_j^{m}(t_m) &\implies i_{m-1} \in S_j^{m}(t) \\
i_m \in S_j^{0}(t_0) &\implies i_m \in S_j^{0}(t)
\end{aligned}
$$

Because $SG_j(t)$, implied by $S_j^i(t)$, is acyclic at every instant t, the above relations indicate a contradiction. Thus circular computation is impossible if the above mentioned

conditions are observed. Notice that we intend to propagate the distances along the shortest-multipath SG_j which is computed using the distances itself. This "bootstrap" approach — computing D_j^i using diffusing computation along SG_j and simultaneously constructing and maintaining SG_j — is the core of MDVA.

2.2.1 Description of MDVA

In essence, MDVA uses DBF to compute D_j^i and therefore SG_j, while always propagating distances along the SG_j to prevent count-to-infinity problem and ensure termination. Each node maintains a *main table* that stores D_j^i, the successor set S_j^i, the feasible distance FD_j^i, the reported distance RD_j^i, and SD_j^i, which is the shortest distance possible through the successor set S_j^i. The table also stores $WN_j^i \subseteq S_j^i$, the set of waiting neighbors in a diffusing computation. Each node also maintains a *neighbor table* for each neighbor k that contains D_{jk}^i the distance of neighbor k to j as communicated by k. The *link table* stores the cost l_k^i of adjacent link to each neighbor k. At startup time, a node initializes all distances in its tables to infinity and all successor sets to null. If a link is down its cost is considered infinity. The distance to unreachable nodes are considered to be infinity.

Nodes executing MDVA exchange information using messages which can have one or more entries. An entry or distance vector is of the form [*type*, j, d], where d is the distance of the node sending the message to destination j and the *type* is one of QUERY, UPDATE and REPLY. We assume that messages transmitted over an operational link are received without errors and in the proper sequence and are processed in the order received.

Nodes invoke the procedure $ProcessDistVect$ shown in Figure 2.1 to process distance vectors. An event is the arrival of a message, the change in cost of an adjacent link, or a change in status (up/down) of an adjacent link. When an adjacent link becomes available,

the node sends an update message [UPDATE, j, RD_j^i] for each destination j over the link. When an adjacent link (i, m) fails, the neighbor table associated with neighbor m is cleared and the cost of the link is set to infinity, after which, for each destination the procedure $ProcessDistVect(UPDATE, m, \infty, j)$ is invoked. Similarly, when an adjacent link cost to m changes, l_m^i is set to the new cost and $ProcessDistVect$(UPDATE, m, D_{jm}^i, j) is invoked for each destination j. When a message is received from neighbor k, $ProcessDistVect(type, k, d, j)$ is invoked for each entry $[type, j, d]$ of the message.

Computing distances to each destination can be performed independently. Hence, in the rest of the description, the working of the algorithm is described with respect to a particular destination j. A node can be in ACTIVE or PASSIVE state with respect to a destination j and is represented by variable $state_j^i$. A node is in ACTIVE state when it is engaged in a diffusing computation and waiting for replies from neighbors. Initially, we assume that all nodes are in PASSIVE state. As long as link cost decrease, MDVA works identically to DBF and the nodes will remain in PASSIVE state. This is because the condition on line 9 always fails and lines 17-24 are always executed. $ProcessDistVect$ works in such a way that when in PASSIVE state, the condition $D_j^i = FD_j^i = RD_j^i = min\{D_{jk}^i + l_k^i | k \in N^i\}$ always holds, which can be inferred from lines 8 and 23. However, if the distance to a destination increases, either because an adjacent link cost changed or a message is received from a neighbor, the condition on line 9 succeeds and the node engages in a diffusing computation. A diffusing computation is initiated by sending query messages to all the neighbors with the best distance SD_j^i through S_j^i, and waiting for the neighbors to reply (lines 14-15). If the increase in distance is due to a query from a successor, the neighbor is added to WN_j^i to indicate that it is waiting for a reply so that a reply can be given when the node transitions to PASSIVE state (lines 11-12). When all replies are received, the node can be sure that the neighbors have incorporated the distances

```
00. procedure ProcessDistVect(et, m, d, j)
01. { et is the type, m is the neighbor, d is the distance, j is the destination }
02. begin
03.     if (j = thisNode ∧ et = QUERY) then send [REPLY, j, 0] to m; endif
04.     D^i_{jm} = d;
05.     D^i_j ← min{D^i_{jk} + l^i_k | k ∈ N^i};
06.     SD^i_j ← min{D^i_{jk} + l^i_k | k ∈ S^i_j};
07.     if (state^i_j = PASSIVE ∨ state^i_j = ACTIVE ∧ last reply is received for j) then
08.         FD^i_j ← min{D^i_j, RD^i_j};
09.         if (D^i_j > RD^i_j) then
10.             state^i_j ← ACTIVE;
11.             if (et = QUERY) then
12.                 WN^i_{jm} ← m;
13.             endif
14.             RD^i_j ← SD^i_j;
15.             ∀k ∈ N^i, send [QUERY, j, RD^i_j] to neighbor k;
16.         else
17.             state^i_j ← PASSIVE;
18.             foreach k ∈ N^i do
19.                 if (k ∈ WN^i_j ∨ (k = m ∧ et = QUERY)) then send [REPLY, j, D^i_j] to k;
20.                 else if (RD^i_j ≠ D^i_j) send [UPDATE, j, RD^i_j] to k;
21.                 endif
22.             done
23.             RD^i_j ← D^i_j;
24.             WN^i_j ← φ;
25.         endif
26.     else
27.         if (et = QUERY) then
28.             if (m ∈ S^i_j ∧ SD^i_j > RD^i_j) then WN^i_j ← WN^i_j ∪ m;
29.             else send [REPLY, j, RD^i_j] to m;
30.             endif
31.         endif
32.     endif
33.     S^i_j ← {k | D^i_{jk} < FD^i_j};
34. end
```

Figure 2.1: Distance vector processing in MDVA.

that the node reported, and is safe to transit to PASSIVE state. At this point, FD^i_j can be increased and new neighbors can be added to S^i_j without violating the LFI conditions.

When in ACTIVE state, if a query message is received from a neighbor *not* in S^i_j, a reply is given immediately. On the other hand, if the query is from a neighbor m in S^i_j, a test is made to verify if SD^i_j increased beyond the previously reported distance (line 28). If it did not, a reply is sent immediately. However, if SD^i_j increased, no reply is given and

the query is blocked by adding m to WN_j^i. The replies to neighbors in WN_j^i are deferred until that time when the node is ready to transit to PASSIVE state. After receiving all replies, one of two things can happen: either the ACTIVE phase ends or it continues. If the distance D_j^i increased again after receipt of all replies, the ACTIVE phase is extended by sending new set of queries, otherwise the ACTIVE phase ends. In the case the ACTIVE phase continues, no replies are issued to the pending queries in WN_j^i. Otherwise, all replies are given and the node transitions to PASSIVE state satisfying the PASSIVE-state invariant $D_j^i = FD_j^i = RD_j^i = min\{D_{jk}^i + l_k^i | k \in N^i\}$.

2.2.2 Correctness Proof of MDVA

To prove the correctness of MDVA consider the following two mutually exclusive and exhaustive cases: (1) some link costs change, but the distances to destinations either decrease or remain unchanged, (2) some link costs increase, resulting in an increase in distances to some destinations. MDVA works identical to DBF when distances to destinations only decrease and the same proof of DBF applies [6]. To state this formally, assume the network is stable up to time t and all nodes have the correct distances. At time t, the costs of some links decrease. Since the distances in the tables are such that $D_j^i(t) \geq \mathbf{D}_j^i(t)$, within some finite time t', $t \leq t' < \infty$, $D_j^i(t') = \mathbf{D}_j^i(t)$.

MDVA and DBF behave differently, when some link costs increase such that distances between some source-destination pairs increase. In this case, $D_j^i(t) < \mathbf{D}_j^i(t)$ for some i and j. Both DBF and MDVA first increase D_j^i to a value greater than $\mathbf{D}_j^i(t)$, after which the distances monotonically decrease until they converge to the correct distances. MDVA and DBF, however, differ on *how* they increase the distances. DBF does it step-by-step in small bounded increments until $D_j^i \geq \mathbf{D}_j^i(t)$. However, when $\mathbf{D}_j^i(t) = \infty$, this leads to the count-to-

infinity problem. In contrast, MDVA uses diffusing computations to quickly raise D_j^i so that $D_j^i \geq \mathbf{D}_j^i(t)$, after which it functions similar to scenario 1 described above, and the distances converge to the correct values as before. After the end of all diffusing computations MDVA works just like DBF.

In summary, to show that MDVA terminates, it is sufficient to show that: (1) the SG_j are loop-free at every instant (Theorem 2), (2) every diffusing computation completes within a finite time (Theorem 3), and (3) there is a finite number of diffusing computations (Theorem 4). Finally, we show that MDVA converges to correct distances when it terminates in Theorem 5.

Theorem 2 Safety Property *For a given destination j, the SG_j constructed by MDVA is loop-free at every instant.*

Proof: The proof is by showing that the LFI conditions are satisfied during every ACTIVE and PASSIVE phase. Let t_n be the time when the n^{th} transition from PASSIVE to ACTIVE state starts at node i for j. The proof is by induction on t_n. At node initialization time 0, all distance variables are initialized to infinite and hence $FD_j^i(0) \leq D_{jk}^i(0)$, $k \in N^i$. Assume the LFI conditions are true up to time t_n. Then

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [0, t_n). \tag{2.5}$$

At any time t, from lines 6, 8, 14 and 23 in the pseudo code in Figure 2.1, and because $SD_j^i(t) \geq D_j^i(t)$, if follows that

$$FD_j^i(t) \leq RD_j^i(t) \tag{2.6}$$

and therefore, for t_{n-1} and t_n, we have

$$FD_j^i(t_{n-1}) \leq RD_j^i(t_{n-1}), \tag{2.7}$$

$$FD_j^i(t_n) \leq RD_j^i(t_n). \tag{2.8}$$

Let the queries sent at t_n, the start time of the n^{th} ACTIVE phase, be received at a particular neighbor k at $t' > t_n$. From Eq. (2.6) and the fact that the update messages sent, if any, between t_{n-1} and t_n specify non-increasing distances, we have

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [t_n, t']. \tag{2.9}$$

Let t'' be the time when all replies are received and ACTIVE phase ends. During the ACTIVE phase the value of FD_j^i remains unchanged and no new RD_j^i is reported during this period (lines 27-31). Furthermore, during PASSIVE phase, only decreasing values of RD_j^i are reported. Then from Eq. (2.8) it follows that

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [t', t''). \tag{2.10}$$

At t'', irrespective of whether the node transitions to PASSIVE state or continues in the ACTIVE phase, from Eq. (2.6) we have

$$FD_j^i(t'') \leq RD_j^i(t''). \tag{2.11}$$

In the case that the ACTIVE phase finally ends, we have $FD_j^i(t) \leq D_{jk}^i(t)$ for $t \in [t_n, t'']$. In the PASSIVE phase, RD_j^i can only remain constant or decrease until the next ACTIVE phase at t_{n+1}. Therefore, the LFI conditions are satisfied in the interval $[t_n, t_{n+1})$. On the other hand, if the ACTIVE phase continues, new queries are sent at time t''. Assume all replies

for these queries are received at time t'''. From similar argument as above, it follows that $FD^i_j(t) \leq D^i_{jk}(t)$ for $t \in [t_n, t''']$. Thus irrespective of how long the ACTIVE phase continues, the invariant holds between $[t_n, t_{n+1}]$. From induction, therefore, the LFI conditions hold at all times. It then follows from Theorem 1 that SG_j is loop-free at all times. □

Theorem 3 *Every ACTIVE phase has a finite duration.*

Proof: An ACTIVE phase may never end due to either of the two reasons: *deadlock* or *livelock*. First we show a deadlock cannot occur. A node that transitions to ACTIVE state with respect to a destination sends queries. If the transition is due to a query from a successor, the node defers the reply to this query until it receives the replies to its own queries. Because nodes wait for replies to their queries before replying to a query, there is a possibility of "circular" waits leading to a deadlock. But, this is impossible for the following reasons. First, a node in passive state immediately replies to a query if it does not increase distance to the destination (lines 19). If the query is from a successor that potentially increases SD^i_j, and the node is ACTIVE , the query is held until the ACTIVE phase ends (line 28). Because the SG_j's are loop-free at every instant (Theorem 2), a deadlock cannot occur. Thus, a node that issued queries to the neighbors will eventually receive all the replies and transitions to PASSIVE state.

A livelock is a situation where a node endlessly has back-to-back ACTIVE phases without ever replying to the pending queries from the successors. A livelock cannot occur for the following reasons. An ACTIVE phase transition occurs either because the link-cost of an adjacent link increases or a query from a successor is received that increases SD^i_j. But, we know that a query from a successor is blocked if it increases SD^i_j. Because links can change only a finite number of times and there is only a finite number of neighbors for each node from which the node can receive queries, the node can only have finite number of back to back active

phases. A node eventually sends all pending replies and enters PASSIVE state. A livelock, therefore, cannot occur. □

Theorem 4 *A node can have only a finite number of ACTIVE phases.*

Proof: Assume towards a contradiction that there is a node that does go through an infinite number of PASSIVE to ACTIVE transitions. An active phase transition occurs either because of a query from a successor or a link-cost increase of an adjacent link. Because link costs can change only a finite number of times, the infinite PASSIVE-ACTIVE phase transitions must have been triggered by an infinite number of queries from a neighbor. Let that neighbor be k. Now, by the same argument, k is sending an infinite number of queries because it is receiving an infinite number of queries. But this argument cannot be continued for ever because there is only a finite number of nodes in the network. Because the reply to the neighbor in the successor set causing the phase transition is blocked and the routing graphs are loop-free at every instant (Theorem 2), there must be a node that transitions to ACTIVE state *only* because of adjacent link cost changes. This implies that a link must change its cost infinite number of times — a contradiction of assumption. Therefore, a node cannot have an infinite number of ACTIVE phases. □

Theorem 5 Liveness Property *After a finite sequence of link-cost changes in the network, the distances D_j^i converge to the final correct values.*

Proof: Assume at time 0 that every node has correct distances to all the distances. In other words, $D_j^i(0) = \mathbf{D}_j^i(0)$. Assume that a finite number of link cost changes, link failures and link recoveries occur in the network between time 0 and t_c and after t_c no more changes occur. We have to show that at some time t_f, such that $t_c \leq t_f < \infty$, all nodes will converge to the correct distances. That is $D_j^i(t_f) = \mathbf{D}_j^i(t_c) = \mathbf{D}_j^i(t_f)$.

From Theorem 3 and 4, it follows that within a finite time after the last link change, all nodes transit to PASSIVE state and remain in PASSIVE state thereafter. Therefore, let t' be the time when the last ACTIVE phase ends in the network. We prove the following.

1. $D_j^i(t') \geq \mathbf{D}_j^i(t_c)$ for every i and j.

2. Between t' and t_f, all D_j^i's monotonically decrease and eventually converge to the correct distances $\mathbf{D}_j^i(t_c)$ at t_f. That is $D_j^i(t_f) = \mathbf{D}_j^i(t_c)$.

Part 1: Assume towards a contradiction that $D_j^i(t') < \mathbf{D}_j^i(t_c)$. Let $D_j^i(t') = (l_k^i(t') + D_{jk}^i(t'))$ for some $k \in K \subseteq N^i$. Assume that $D_j^k(t') \geq \mathbf{D}_j^k(t_c)$. Also assume that K has only one element. Because $\mathbf{D}_j^i(t_c) = \mathbf{l}_k^i(t_c) + \mathbf{D}_j^k(t_c)$ we have $l_k^i(t') + D_{jk}^i(t') \leq \mathbf{l}_k^i(t_c) + D_j^k(t')$, from which we can infer that either $l_k^i(t') < \mathbf{l}_k^i(t_c)$, or $D_{jk}^i(t') < D_j^k(t')$, or both. If $l_k^i(t') < \mathbf{l}_k^i(t_c)$, it implies that the link cost of (i,k) is not yet increased to $\mathbf{l}_k^i(t_c)$ via a link-cost change event. When it does, the condition on line 9 becomes true and an ACTIVE state transition is triggered. So all ACTIVE phases have not ended. Similarly, if $D_{jk}^i(t') < D_j^k(t')$, then there is message in transit, which when processed by i would trigger a PASSIVE-to-ACTIVE transition. This means that the ACTIVE phases have not yet ended. A contradiction of the assumption. Therefore, when ACTIVE phases end $D_j^i(t') \geq \mathbf{D}_j^i(t_c)$. When K has more than one element, each element will be removed from the successor set one after the other without triggering the ACTIVE transition until the last element, when the ACTIVE state transition finally occurs.

Part 2: After every node becomes PASSIVE at time t', all the messages in transit can only decrease the distances; otherwise, that would result in a transition to ACTIVE state. At this stage MDVA works essentially like DBF and the same proof of DBF applies here. Each time a distance is decreased, the new distance is reported. Because distances cannot decrease forever and are lower bounded by $\mathbf{D}_j^i(t_c)$, the distances will eventually converge to the correct distances $\mathbf{D}_j^i(t_c)$. □

```
00. procedure PDA
01. {Executed at each router i. Invoked when an event occurs}
02. begin
03.       call NTU-PDA;
04.       call MTU-PDA; /* Updates T^i */
05.       if (there are changes to T^i) then
06.            Compose an LSU message M specifying changes to T^i
07.            using add, delete and change link entries;
08.       endif
09.       Within a finite amount time, send M to all neighbors;
10. end PDA
```

Figure 2.2: The Partial-topology Dissemination Algorithm

2.3 Multipath Link State Algorithm

We present the *Multipath Partial Dissemination Algorithm* (MPDA) in two steps. We first describe the *Partial Dissemination Algorithm* (PDA) for computing the shortest distances to destinations. We then extend PDA to incorporate the LFI conditions to obtain MPDA. The PDA is a shortest path routing algorithm in its own right and can be used in practice, but its advantage here is that it can be modified easily to enforce LFI conditions.

2.3.1 Description of PDA

The shortest-path routing algorithm, PDA, propagates enough link-state information in the network, so that each router has *sufficient* link-state information to compute shortest paths to all destinations. In this respect, it is similar to other link-state algorithms (e.g., OSPF [45], SPTA [59], LVA [31], ALP [33]). PDA combines the best features of LVA, ALP and SPTA. As in LVA and ALP, a router communicates to its neighbors information regarding only those links that are part of its minimum-cost routing tree, and like SPTA, a router validates link information based on distances to heads of links and not on sequence numbers.

PDA assumes that a router detects the failure, recovery and link-cost change of an adjacent link within a finite amount of time. An underlying protocol ensures that messages

```
00. procedure NTU-PDA
01. begin
02.     if (LSU message is received from a neighbor k) then
03.         Update neighbor table T_k^i; /* i.e., add links,
04.             delete links or change links according to the
05.             specification of each entry in the LSU. */
06.         Run Dijkstra's shortest path algorithm
07.             on the resulting topology T_k^i; /* This gives
08.             minimum distances from k to all other
09.             nodes in T_k^i. Note T_k^i is a tree. */
10.         Update D_jk^i with new distances in T_k^i;
11.     endif
12.     if (adjacent link (i, k) is up) then
13.         Update l_k^i;
14.         Send an LSU message to k with link information
15.         of all links in its main topology table T^i;
16.     endif
17.     if (cost of an adjacent link (i, k) changed) then
18.         Update l_k^i;
19.     endif
20.     if (adjacent link (i, k) failed) then
21.         Update l_k^i and clear the table T_k^i;
22.     endif
23. end NTU-PDA
```

Figure 2.3: Neighbor topology table update procedure in PDA

transmitted over an operational link are received correctly and in the proper sequence within a finite time and are processed by the router one at a time in the order received. These are the same assumptions made for similar routing algorithms and can be easily satisfied in practice. Each router i running PDA maintains the following information:

1. The *main topology table*, T^i, stores the characteristics of each link known to router i. Each entry in T^i is a triplet $[h, t, d]$ where h is the head, t is the tail and d is the cost of the link $h \rightarrow t$.

2. The *neighbor topology table*, T_k^i, is associated with each neighbor k. The table stores the link-state information communicated by the neighbor k. That is, T_k^i is a time-delayed version of T^k.

```
00. procedure MTU-PDA
01. begin
02.     oldT^i ← T^i;/* Save copy */
03.     if (node j occurs in at least one of T_k^i) then
04.         Add j to the main topology table T^i;
05.     endif
06.     foreach node j in T^i do
07.         MIN ← min{D_jk^i + l_k^i | k ∈ N^i};
08.         let p be such that MIN = (D_jp^i + l_p^i);
09.         /* Neighbor p is the preferred neighbor for
10.         destination j. Ties are broken in favor of
11.         lower address neighbor */
12.     done
13.     foreach j in T^i and its preferred neighbor p do
14.         Copy all links (j, n) from T_p^i to T^i;
15.         /* i.e., copy all links in T_p^i for which
16.         j is the head node. */
17.     done
18.     Update T^i with information of each l_k^i;
19.     Run Dijkstra's shortest path algorithm on T^i
20.         and remove those links in T^i that are not
21.         part of the shortest path tree;
22.     Update D_j^i with new distances in T^i;
23.     Compare oldT^i with T^i and note all differences;
24. end MTU-PDA
```

Figure 2.4: Main topology table update procedure in PDA

3. The *distance table* stores the distances from router i to each destination based on the topology in T^i and the distances from each neighbor k to each destination based on the topologies in T_k^i for each k. The distance of router i to node j in T^i is denoted by D_j^i; the distance from k to j in T_k^i is denoted by D_{jk}^i.

4. The *routing table* stores, for each destination j, the successor set S_j^i and the feasible distance FD_j^i, which is used by MPDA to enforce LFI conditions.

5. The *link table* stores, for each neighbor k, the cost l_k^i of the adjacent link to the neighbor.

The unit of information exchanged between routers is a link-state update (LSU) message. A router sends an LSU message containing one or more entries, with each entry specifying *addition*, *deletion* or *change* in cost of a link in the router's main topology table

T^i. Each entry of an LSU consists of link information in the form of a triplet $[h, t, d]$ where h is the head, t is the tail, and d is the cost of the link $h \rightarrow t$. An LSU message contains an acknowledgment (ACK) flag for acknowledging the receipt of an LSU message from a neighbor (used only by MPDA).

When a router is powered up, it initializes the tables; all variables of type distance are initialized to infinity and those of type node are initialized to null. All successor sets are initialized to the empty set. The PDA in Fig. 2.2 is then invoked. PDA is executed each time an event occurs; an event can be either a receipt of an LSU message from a neighbor or the detection of an adjacent link-cost change. Procedure NTU-PDA shown in Fig. 2.3 is used to process the received message and update the necessary tables. Procedure MTU-PDA in Fig. 2.4 constructs the router's own shortest path tree from the topologies reported by its neighbors. The new shortest-path tree obtained is compared with the previous version to determine the differences; only the differences are then reported to the neighbors. The router then waits for the next event and, when it occurs, the whole process is repeated.

The algorithm MTU-PDA at router i merges the topologies T_k^i and the adjacent links l_k^i to obtain T^i. The merge process is straightforward if all neighbor topologies contain disjoint sets of links, but when two or more neighbors report conflicting information regarding a particular link, the conflict has to be resolved. Sequence numbers may be used to distinguish between old and new link information as in OSPF, but PDA resolves the conflict as follows. If two or more neighbors report information of link $(m,\ n)$ then the router i should update topology table T^i with link information reported by the neighbor that offers the shortest distance from the router i to the head node m of the link. Ties are broken in favor of neighbor with the lowest address. For adjacent links, router i itself is the head of the link and thus has the shortest distance. Therefore, any information about an adjacent link supplied by

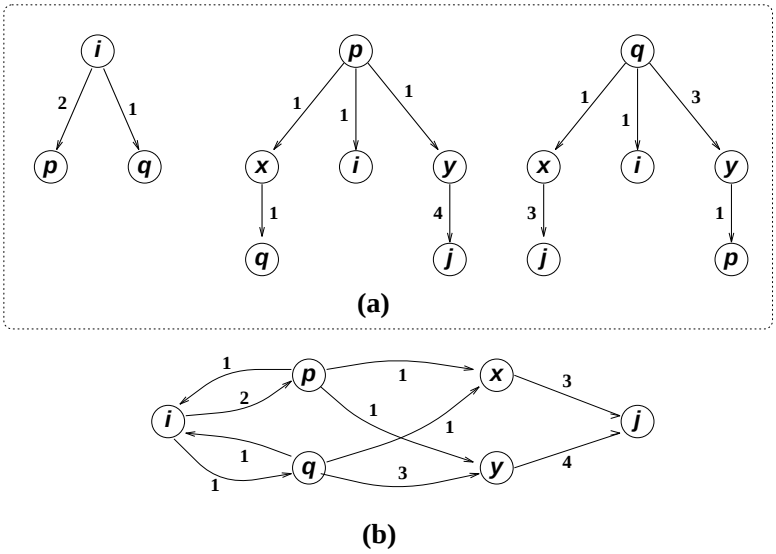


Figure 2.5: Illustration of the main table update procedure of PDA.

Table 2.1: Preferred neighbor table

Destination	p	q	x	y	j
Distance	2	1	2	3	5
Pref. Nbr	p	q	q	p	q

neighbors will be overridden by the most current information about the link available to router i. Dijkstra's shortest path algorithm is applied on T^i, with i as the root and only the links that constitute the shortest-path tree are retained. Note that, because there are potentially many shortest-path trees, ties should be broken consistently during the run of Dijkstra's algorithm. As an example consider Fig. 2.5. Fig. 2.5(a) shows the main table of i and the neighbor tables for p and q at node i. The distances to various destinations and the preferred neighbors is given in Table 2.1. After merging the main link table and neighbor tables, the resulting topology is shown in Fig. 2.5(b).

To see why breaking ties consistently is important, consider the network in Fig. 2.6(a). Fig. 2.6(b) shows the adjacent links and the shortest path trees of its neighbors p and q. The

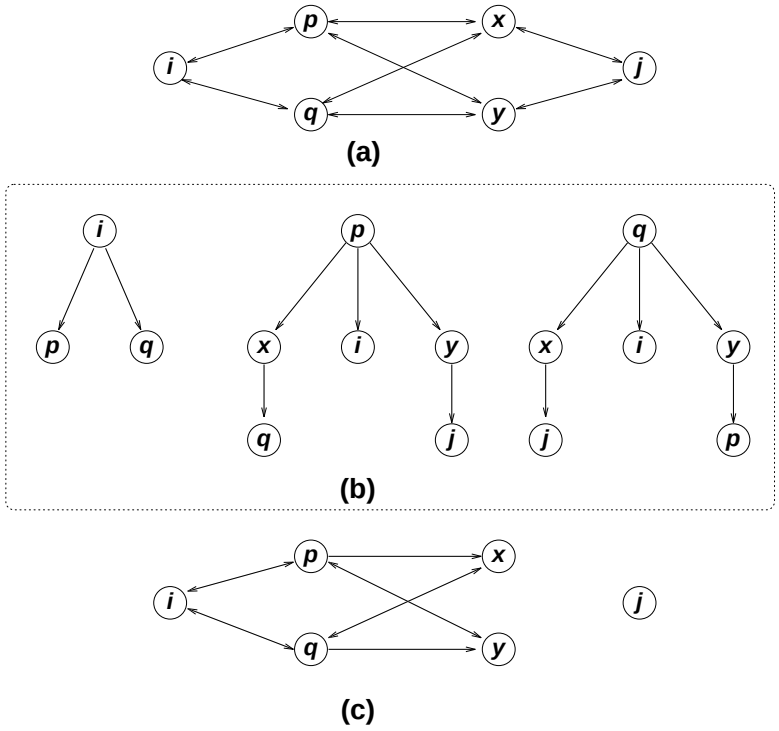


Figure 2.6: Illustrating the significance of the tie-breaking rule.

distances of nodes x and y from i is identical through both neighbors p and q. Now if MTU-PDA breaks ties in arbitrary manner while constructing T^i, it may choose p as the preferred neighbor for node x and choose q as preferred neighbor for node y, resulting in a graph, as shown in Fig. 2.6(c), that has no path from i to j. Ties, therefore, cannot be broken in arbitrary manner.

2.3.2 Correctness Proof of PDA

In what follows, we show that PDA works correctly by showing that the topology tables at all nodes converge to the shortest paths within a finite time after the last link cost change in the network. After convergence, because there are no more changes to the topology

tables, no more LSU messages are generated.

Definitions: The n-hop minimum distance of router i to node j in a network is the minimum distance possible using a path of n links or less. A path that offers the n-hop minimum distance is called n-hop minimum path. If there is no path with n hops or less from router i to j then the n-hop minimum distance from i to j is undefined. An n-hop minimum tree of a node i is a tree in which router i is the root and all paths of n hops or less from the root to any other node is an n-hop minimum path. Note that there could be more than one n-hop minimum tree.

Let $\mathbf{G}$ denote the final topology of the network after all link changes occurred as seen by an omniscient observer; we use bold font to refer to all quantities in $\mathbf{G}$. Let $\mathbf{H}_n^i$ denote an n-hop minimum tree rooted at router i in $\mathbf{G}$ and let $\mathbf{M}_n^i$ be the set of nodes that are within n hops from i in $\mathbf{H}_n^i$. Let $\mathbf{D}_n^{i,j}$ denote the distance of i to j in $\mathbf{H}_n^i$. Let $\mathbf{d}_{ij}$ be the cost of the link $i \rightarrow j$. The notation $i \rightsquigarrow j$ indicates a path from i to j of zero or more links.

Property 1 *From the principle of optimality (a sub-path of a shortest path between two nodes is also a shortest path between the end nodes of the sub-path), if H and H' are two n-hop minimum trees rooted at router i and M and M' are sets of nodes that are within n hops from i in H and H' respectively, then $M = M' = \mathbf{M}_n^i$. Also, for each $j \in \mathbf{M}_n^i$ the length of path $i \rightsquigarrow j$ in both H and H' is equal to $\mathbf{D}_n^{i,j}$. Also, $\mathbf{D}_h^{i,j} \leq \mathbf{D}_n^{i,j}$ if $h \geq n$.*

We say a router i *knows at least* the n-hop minimum tree, if the tree represented by its main topology table T^i is at least an n-hop minimum tree rooted at i in $\mathbf{G}$ and there are at least n nodes in T^i that are reachable from the root i. Note that the links in T^i that are more than n hops may have costs that do not agree with the link costs in $\mathbf{G}$.

Lemma 1 *If a router i has the final correct costs of the adjacent links and for each neighbor k the topology T_k^i is an n-hop minimum tree, then the topology T^i is $(n+1)$-hop minimum tree*

after the execution of MTU-PDA.

Proof: Let $A^i = \bigcup_{k \in N^i} A^i_k$ where A^i_k is the set of nodes in T^i_k. Since T^i_k is at least a $(n-1)$-hop minimum tree and node i can appear at most once in each of A^i_k, each A^i_k has at least $n-1$ unique elements. Therefore A^i has at least $n-1$ elements.

Let M^i_n be the set of $n-1$ nearest elements to node i in A^i. That is $M^i_n \subseteq A^i$ and $|M^i_n| = n-1$ and for each $j \in M^i_n$ and $v \in A^i - M^i_n$, $min\{D^i_{jk} + l^i_k | k \in N^i\} \leq min\{D^i_{vk} + l^i_k | k \in N^i\}$.

The theorem is proved in the following two parts:

1. Let G^i_n represent the graph constructed by MTU-PDA on line 13-18. (i.e., before applying Dijkstra on line 19-21). For each $j \in M^i_n$ there is a path $i \rightsquigarrow j$ in G^i_n such that its length is at most $\mathbf{D}^{i,j}_n$.

2. After running Dijkstra on G^i_n on line 19-21 in MTU-PDA, the resulting tree is at least an n-hop minimum tree.

Let us first assume Part 1 is true and prove Part 2, and then proceed to prove Part 1. From the statement in Part 1, for each node $j \in M^i_n$ there is a path $i \rightsquigarrow j$ in G^i_n with length at most $\mathbf{D}^{i,j}_n$. After running Dijkstra's algorithm, in the resulting graph, we can infer that there is a path $i \rightsquigarrow j$ with length at most $\mathbf{D}^{i,j}_n$. Because there are $n-1$ nodes in M^i_n, the tree constructed has at least n nodes with node i included. Accordingly, it follows from Property 1 that the tree constructed is at least an n-hop minimum tree.

Now we prove Part 1. Order the nodes in M^i_n in non-decreasing order. The proof is by induction on the sequence of elements in M^i_n as they are added to G^i_n. The base case is when G^i_n contains just one link $l^i_{m_1} = min\{l^i_k | k \in N^i\}$ and m_1 is the first element of M^i and $l^i_{m_1} = \mathbf{D}^{i,m_1}_1$. Let the statement hold for the first $m-1$ elements of M^i_n and consider the m-th element $j \in M^i_n$.

Let K be the highest priority neighbor for which $D^i_{jK} + l^i_K = min\{D^i_{jk} + l^i_k | k \in N^i\}$. At Most $m-2$ nodes in T^i_K can have a smaller or equal distance than j, which implies path $K \rightsquigarrow j$ exists with at most $m-1$ hops. Let v be the neighbor of j in T^i_K. Then the path $K \rightsquigarrow v \rightarrow j$ has at most $m-1$ hops. Because T^i_K is at least a $(n-1)$-hop minimum tree, the cost of link $v \rightarrow j$ must agree with $\mathbf{G}$. Since $D^i_{vK} + l^i_K < D^i_{jK} + l^i_K$, from our inductive hypothesis , there is a path $i \rightsquigarrow v$ in G^i_n such that the length is at most $\mathbf{D}^{i,v}_n$.

Now we need to show that the preferred neighbor for v is also K, so that the link $v \rightarrow j$ will be included in the construction of G^i_n, thus ensuring the existence of the path $i \rightsquigarrow j$ in G^i_n. If some other neighbor K' instead of K is the preferred neighbor for v, then one of the following two cases should have occurred: (a) $D^i_{vK'} + l^i_{K'} < D^i_{vK} + l^i_K$ or, (b) $D^i_{vK'} + l^i_{K'} = D^i_{vK} + l^i_K$ and priority of K' is greater than priority of K.

Case (a): If $D^i_{vK'} + l^i_{K'} < D^i_{vK} + l^i_K$, then given that $D^i_{jK} + l^i_K \leq D^i_{jK'} + l^i_{K'}$ it follows that the path $v \rightsquigarrow j$ in $T^i_{K'}$ is greater than cost $v \rightarrow j$ in $\mathbf{G}$ which implies that $T^i_{K'}$ is not a $(n-1)$ hop minimum tree – a contradiction to our assumption! Therefore, $D^i_{vK} + l^i_K = min\{D^i_{vk} + l^i_k | k \in N^i\}$.

Case (b): Let Q_j be the set of neighbors that give the minimum distance to j, i.e., for each $k \in Q_j$, $D^i_{jk} + l^i_k = min\{D^i_{jk} + l^i_k | k \in N^i\}$. Similarly, let Q_v be such that for each $k \in Q_v$, $D^i_{vk} + l^i_k = min\{D^i_{vk} + l^i_k | k \in N^i\}$. If $k \in Q_v$ and $k \notin Q_j$, then it follows from the same argument used in case (a) that $v \rightsquigarrow j$ in T^i_k is greater than $v \rightarrow j$ in $\mathbf{G}$, which implies that T^i_k is not a $(n-1)$-hop minimum tree – a contradiction to our assumption again. Therefore, $Q_v \subseteq Q_j$. Also, from the same argument used in case (a) above it can be inferred that $K \in Q_v$. Because K has the highest priority among all members of Q_j and $Q_v \subseteq Q_j$, and because $k \in Q_v$, K must also have the highest priority among all members of Q_v. This proves that $v \rightarrow j$ will be included in the construction of G^i_n. Because $\mathbf{D}^{i,v}_n + \mathbf{d}_{vj} = \mathbf{D}^{i,j}_n$ in $\mathbf{G}$, where

$\mathbf{d}_{vj}$ is the final cost of link $v \to j$, and the length of $i \rightsquigarrow v$ in G_n^i is less than $\mathbf{D}_n^{i,v}$ from our inductive hypothesis, we obtained that the length of $i \rightsquigarrow j$ in G_n^i less than $\mathbf{D}_n^{i,j}$. This proves Part 1 of the theorem. □

Theorem 6 *At each router i, the main topology T^i gives the correct shortest paths to all known destinations a finite time after the last change in the network.*

Proof: The proof is by induction on t_n, the global time when for each router i, T^i is at least n-hop minimum tree. Because the longest loop-free path in the network has at most $N-1$ links where N is number of nodes in the network, t_{N-1} is the time when every router has the shortest path to every other node. We need to show that t_{N-1} is finite. The base case is t_1, the time when every node has 1-hop minimum distance and because the adjacent link changes are notified within finite time, $t_1 < \infty$. Let $t_n < \infty$ for some $n < N$. Given that the propagation delays are finite each router will have each of its neighbors n-hop minimum tree in finite time after t_n. From Theorem 1 we can see that the router will have at least the $(n+1)$-hop minimum tree within a finite time after t_n. Therefore, $t_{n+1} < \infty$. From induction, we can conclude that $t_{N-1} < \infty$. □

2.3.3 Description of MPDA

The LFI conditions introduced in Section 2.1 suggest a technique for computing S_j^i such that the implied routing graph SG_j is loop-free at every instant. To determine FD_j^i in Eq.(2.1), router i needs to know D_{ji}^k, the distance from i to node j in the topology table T_i^k. Because of propagation delays, there may be discrepancies between the main topology table T^i at router i and its copy T_i^k at the neighbor k. However, at time t, the topology table T_i^k is a copy of the main topology table T^i at some earlier time $t' < t$. Logically, if a copy of D_j^i is saved each time an LSU is sent, a feasible distance FD_j^i that satisfies the LFI conditions can

```
00. procedure MPDA at router i
01. {invoked when an event occurs}
02. begin
03.       call NTU-PDA;
04.       if (node is in PASSIVE state) then
05.            call MTU-PDA; /* update T^i and D^i_j */
06.            FD^i_j ← min{FD^i_j, D^i_j};
07.       endif
08.       if (node is in ACTIVE state and the
09.                 last ACK is received) then
10.            temp^i_j ← D^i_j; Set node to PASSIVE state;
11.            call MTU-PDA to update T^i;
12.            FD^i_j ← min{temp^i_j, D^i_j};
13.       endif
14.       S^i_j ← {k|D^i_jk < FD^i_j};
15.       if (changes occur in T^i)then
16.            Set node to ACTIVE state;
17.       endif
18.       if (no changes occur in T^i and the event is
19.                 the last ACK) then
20.            Set node to PASSIVE state;
21.       endif
22.       if (there are changes to T^i) then
23.            Compose a new LSU with the topology
24.                 changes expressed as add link,
25.                 delete link and change link;
26.       endif
27.       if (input event received is an LSU message) then
28.            Add the ACK entry to newly composed LSU;
29.       endif
30.       Send the new LSU message;
31. end MPDA
```

Figure 2.7: Multiple-path Partial-topology Dissemination Algorithm (MPDA)

be found in the history of values of D_j^i that have been saved!

The Multiple-path Partial-Topology Dissemination Algorithm, or MPDA, shown in Fig. 2.7 is a modification of PDA that enforces the LFI conditions by synchronizing the exchange of LSUs between neighbors. In MPDA, each LSU message sent by a router is acknowledged by all its neighbors before the router sends the next LSU. The inter-neighbor synchronization used in MPDA spans only a *single* hop, unlike the synchronization in diffusing computations [27] which potentially spans the whole network. A router is said to be in

ACTIVE state when it is waiting for its neighbors to acknowledge the LSU message it sent; otherwise, it is in PASSIVE state.

Assume that, initially, all routers are in PASSIVE state with all routers having the correct distances to all destinations. Then a series of link cost changes occurs in the network resulting in some or all routers to go through a sequence of PASSIVE-to-ACTIVE and ACTIVE-to-PASSIVE state transitions, until all routers become PASSIVE with correct distances to destinations.

If a router in a PASSIVE state receives an event that does not change its topology T^i, then the router has nothing to report and remains in PASSIVE state. However, if a router in PASSIVE state receives an event that affects a change in its topology, the router sends those changes to its neighbors, goes into ACTIVE state and waits for ACKs. Events that occur during the ACTIVE period are processed to update T_k^i and l_k^i but not T^i; the updating of T^i by MTU is deferred until the end of the ACTIVE phase. At the end of the ACTIVE phase, when ACKs from all neighbors are received, router i updates T^i with changes that may have occurred in T_k^i due to events received during the ACTIVE phase. If no changes occurred in T^i that need reporting, then the router becomes PASSIVE; otherwise, as shown in Fig. 2.8, there are changes in T^i that may have resulted due to events and the neighbors need to be notified. This results in a new LSU, and the router immediately becoming ACTIVE again. In this case, there is an implicit PASSIVE period, of zero length of time, between two back-to-back ACTIVE periods, as illustrated in Fig. 2.8. A router i receiving an LSU message from k must send back an LSU with the ACK bit set after updating T_k^i. If the router does not have any updates to send, either because it is in ACTIVE state or because it does not have any changes to report, it sends back an empty LSU with just the ACK flag set. When a router detects that an adjacent link failed, any pending ACKs from the neighbor at the other end of the link are

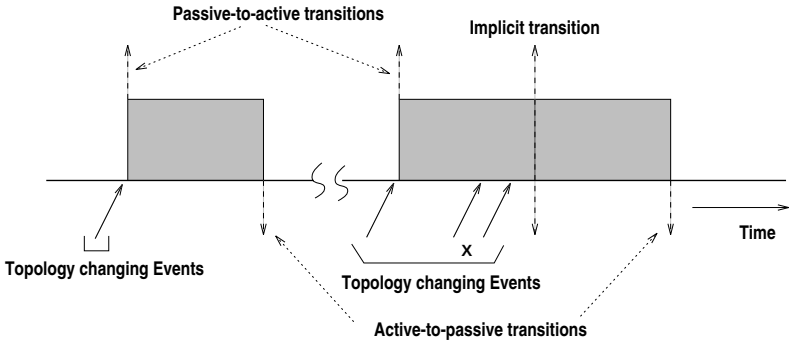


Figure 2.8: Active-passive phase transitions in MPDA.

treated as received. Because all LSUs are acknowledged within a finite time, no deadlocks can occur.

2.3.4 Correctness Proof of MPDA

Theorem 7 *(Safety property) At any time t, the directed graph $SG_j(t)$ implied by the successor sets $S_j^i(t)$ computed by MPDA at each router is loop-free.*

Proof: Let t_n be the time when FD_j^i is updated for the n-th time. The proof is by induction on the time intervals $[t_n, t_{n+1}]$. As inductive hypothesis assume that

$$FD_j^i(t) \quad \leq \quad D_{ji}^k(t) \qquad k \in N^i, t \leq t_n \tag{2.12}$$

We show that

$$FD_j^i(t) \quad \leq \quad D_{ji}^k(t) \qquad t \in [t_n, t_{n+1}] \tag{2.13}$$

We observe from the description of MPDA in Fig. 2.7 that, when FD_j^i is updated at lines 6 and 12, D_j^i is also updated at lines 5 and 11 respectively. We also observed that FD_j^i is updated only during state transitions, and regardless of whether the transition is from PASSIVE-to-ACTIVE or from ACTIVE-to-PASSIVE, the Eq. (2.14) below is true. Note

that there is an implicit PASSIVE state between two back-to-back ACTIVE states.

$$FD_j^i(t_n) \quad \leq \quad min\{D_j^i(t_{n-1}), D_j^i(t_n)\} \tag{2.14}$$

Let t' be the time when LSU sent by i at t_n is received and processed by neighbor k. Because of the non-zero propagation delay across any link, t' is such that $t_n < t' < t_{n+1}$. We then have

$$D_{ji}^k(t') \quad = \quad D_j^i(t_n) \tag{2.15}$$

Because FD_j^i is modified at t_n and then remains unchanged within (t_n, t_{n+1}), we obtain from Eq. (2.12) that

$$FD_j^i(t) \quad \leq \quad D_{ji}^k(t) \qquad t \in [t_n, t') \tag{2.16}$$

From Eqs. (2.14) and (2.15) we obtain the following.

$$FD_j^i(t) \quad \leq \quad D_{ji}^k(t) \qquad t \in [t', t_{n+1}) \tag{2.17}$$

From Eq. (2.16) and (2.17) we have

$$FD_j^i(t) \quad \leq \quad D_{ji}^k(t) \qquad t \in [t_n, t_{n+1}) \tag{2.18}$$

At t_{n+1}, again from the design of MPDA we have,

$$FD_j^i(t_{n+1}) \quad \leq \quad min\{D_j^i(t_n), D_j^i(t_{n+1})\} \tag{2.19}$$

Also, because propagation delays are positive, node k at t_{n+1} cannot yet have the value $D_j^i(t_{n+1})$. So, we have

$$D_{ji}^k(t_{n+1}) \quad = \quad D_j^i(t_n) \tag{2.20}$$

Combining Eq. (2.20) and (2.19) for time t_{n+1}, we get

$$FD_j^i(t_{n+1}) \quad \leq \quad D_{ji}^k(t_{n+1}) \tag{2.21}$$

and Eq. (2.13) follows from combining Eqs. (2.18) and (2.21).

Because $FD_j^i(t_0) \leq D_{ji}^k(t_0)$ at initialization, from induction we have that $FD_j^i(t) \leq D_{ji}^k(t)$ for all t. Given that the successor sets are computed based on FD_j^i, it follows that the LFI conditions are always satisfied. According to Theorem 1, this implies that the successor graph SG_j is always loop-free. □

Theorem 8 *(Liveness property) A finite time after the last change in the network, D_j^i gives the correct shortest distance and*

$$S_j^i = \{k | D_j^k < D_j^i, k \in N^i\} \quad \textit{at each router } i.$$

Proof: The convergence of MPDA follows directly from the convergence of PDA, because the update messages in MPDA are only delayed a finite time as allowed in line 9 in algorithm PDA. Therefore, the distances D_j^i in MPDA also converge to shortest distances. Because changes to T^i are always reported to the neighbors and are incorporated by the neighbors in their tables in finite time, $D_{jk}^i = D_j^k$ for $k \in N^i$ after convergence. From line 10 and 12 in MPDA, we observe that when router i becomes PASSIVE, and $FD_j^i = D_j^i$ holds true. Because all routers are PASSIVE at convergence time it follows that the set $\{k | D_{jk}^i < FD_j^i, k \in N^i\}$ is the same as the set $\{k | D_j^k < D_j^i, k \in N^i\}$. □

2.4 MPATH Routing Algorithm

Most of the routing protocols use either link-states or distance-vectors in their communication. There is another class of protocols that use distance-vectors combined with the identity of the second-to-last node, also called predecessor node, that is just before the destination node on the shortest path. MPATH is the first multipath protocol in this class of routing protocols.

The following information is maintained at each node i:

1. The *Main Distance Table* contains D_j^i and p_j^i, where D_j^i is the distance of node i to destination j and p_j^i is the predecessor to destination j on the shortest path from i to j. The table also stores for each destination j, the successor set S_j^i, feasible distance FD_j^i, reported distance RD_j^i and two flags *changed* and *report-it*.

2. The *Main Link Table* T^i is the node's view of the network and contains links represented by $(m,\ n,\ d)$ where $(m,\ n)$ is a link with cost d.

3. The *Neighbor Distance Table* for neighbor k contains D_{jk}^i and p_{jk}^i where D_{jk}^i is the distance of neighbor k to j as communicated by k and p_{jk}^i is the predecessor to j on the shortest path from k to j as notified by k.

4. The *Neighbor Link Table* T_k^i is the neighbor k's view of the network as known to i and contains link information derived from the distance and predecessor information in the neighbor distance table.

5. *Adjacent Link Table* stores the cost l_k^i of adjacent link to each neighbor k. If a link is down its cost is infinity.

Nodes exchange information using update messages which have the following format.

1. An update message can one or more update entries. An update entry is a triplet [j, d, p], where d is the distance of the node sending the message to destination j and p is the predecessor on the path to j.

2. Each message carries two flags used for synchronization: *query* and *reply*.

```
00. procedure PATH
01. {Invoked when a message M is received from neighbor k,
02. or an adjacent link to k has changed or when a node is
03. initialized.}
04. begin
05.       Run NTU-PATH to update neighbor tables.
06.       Run MTU-PATH to update main tables.
07.       foreach j marked as changed do
08.             Add update entry [j, D_j^i, p_j^i] to the new message M';
09.       done 10.Within finite amount of time, send message M' to
11.       each neighbor.
12. end PATH
```

Figure 2.9: The PATH Algorithm

```
00. procedure NTU-PATH
01. begin
02. {Called by PATH to process an event.}
03. If event is a message M from neighbor k,
04.       foreach ([j, d, p] in M) do /* Note d = D_j^k, p = p_j^k.*/
05.             Set D_jk^i ← d and p_jk^i ← p.
06.       done
07.       foreach (destination j with an entry in M ) do
08.             Remove existing links (n, j) in T_k^i and add new
09.             link (m, j, d) to T_k^i, where d = D_jk^i − D_mk^i
10.             and m = p_jk^i;
11.       done
12.       if (event is an adjacent link-status change) then
13.             Update l_k^i;
14.             Clear neighbor tables of k, if link is down;
15.       endif
16. end NTU-PATH
```

Figure 2.10: Neighbor table update algorithm in PATH

2.4.1 Description of MPATH

As mentioned earlier, our strategy is to first design a shortest-path routing algorithm, PATH, and then make the multipath extensions to it. Figure 2.9 shows the pseudo code of PATH. Note that PATH is essentially the same as PDA in that the internal representation of the links and the update procedure are essentially the same, but differ only in the type of messages exchanged. It node startup time the tables are initialized; distances are initialized

```
00. procedure MTU-PATH
01. begin
02.     Clear link table T^i.
03.     foreach (node j ≠ i occurring in at least one T_k^i) do
04.         Find MIN ← min{D_jk^i + l_k^i | k ∈ N^i};
05.         Let n be such that MIN = (D_jn^i + l_n^i). /* Ties are
06.         broken consistently. Neighbor n is the preferred neighbor
07.         for destination j. */
08.         foreach (link (j, v, d) in T_n^i) do
09.             Add link (j, v, d) to T^i;
10.         done
11.     Update T^i with each link l_k^i;
12.     Run Dijkstra's shortest path algorithm on T^i to
13.         find new D_j^i, and p_j^i;
14.     foreach (destination j) do
15.         if (D_j^i or p_j^i changed from previous value) then
16.             Set changed and report-it flags for j;
17.         endif
18. end MTU-PATH
```

Figure 2.11: Main table update procedure of PATH protocol

to infinity and node identities to a null value. The PATH procedure is then invoked. PATH is again executed in response to an event that can be either a receipt of an update message from a neighbor or detection of an adjacent link cost or link status (up/down) change. PATH invokes procedure NTU-PATH, described in Figure 2.10, which first updates the neighbor distance tables and then updates T_k^i with links $(m,\ n,\ d)$ where $d = D_{nk}^i - D_{mk}^i$ and $m = p_{nk}^i$. PATH then invokes procedure MTU-PATH, specified in Figure 2.11, which constructs T^i by merging the topologies T_k^i and the adjacent links l_k^i.

Now, the final desired routing algorithm MPATH is derived by making extensions to PATH. Pseudo code for MPATH is shown in Figure 2.12. MPATH computes the successor sets S_j^i by enforcing the Loop-Free Invariant conditions described before and using a neighbor-to-neighbor synchronization.

Let FD_j^i, called the feasible distance, be an 'estimate' of the distance of node i to node j in the sense that FD_j^i is equal to D_j^i when the network is in stable state, but to prevent

```
01. procedure MPATH
02. {Invoked when a message M is received from neighbor k,
03. or an adjacent link to k has changed.}
04. begin
05.     Run NTU-PATH to update neighbor tables;
06.     Run MTU-PATH to obtain new D_j^i and p_j^i;
07.     if (node is in PASSIVE state or node is in ACTIVE state ∧ last reply arrived) then
08.         Reset goactive flag;
09.         foreach (destination j marked as report-it) do
10.             FD_j^i ← min{D_j^i, RD_j^i};
11.             if (D_j^i > RD_j^i) then Set goactive flag; endif
12.             RD_j^i ← D_j^i;
13.             Add [j, RD_j^i, p_j^i] to message M';
14.             Clear report-it flag for j;
15.         done
15.     else /* the node is ACTIVE and waiting for more replies */
16.         foreach (destination j marked as changed) do
17.             FD_j^i ← min{D_j^i, FD_j^i};
18.         done
19.     endif
20.     foreach (destination j marked as changed) do
21.         Clear changed flag for j;
22.         S_j^i ← {k|D_jk^i < FD_j^i};
23.     done
24.     foreach (neighbor k ) do
25.         M'' ← M';
26.         If event is a query from k, Set reply flag in M'';
27.         If goactive set, Set query flag in M'';
28.         If M'' non-empty, send M'' to k;
29.         If goactive set, become ACTIVE, otherwise become PASSIVE;
30.     done
31.end MPATH
```

Figure 2.12: Multi-path Loop-free Routing Algorithm

loops during periods of network transitions, it is allowed to be temporarily differ from D_j^i.

The invariants used in LFI are independent of whether the algorithm uses link states or distance vectors; in link-state algorithms, such as MPDA, the D_{jk}^i are computed locally from the link-states communicated by the neighbors while in distance-vector algorithms, like the MPATH presented here, the D_{jk}^i are directly communicated.

The invariants (2.1) and (2.2) suggest a technique for computing $S_j^i(t)$ such that the successor graph $SG_j(t)$ for destination j is loop-free at every instant. The key is determining

$FD_j^i(t)$ in Eq. (2.1), which requires node i to know $D_{ji}^k(t)$, the distance from i to node j in the topology table T_i^k that node i communicated to neighbor k. Because of non-zero propagation delay, T_i^k is a time-delayed version of T^i. We observe that, if node i delays updating of FD_j^i with D_j^i until k incorporates the distance D_j^i in its tables, then FD_j^i satisfies the LFI condition.

MPATH enforces the LFI conditions by synchronizing the exchange of update messages among neighbors using *query* and *reply* flags. If a node sends a message with a *query* bit set, then the node must wait until a *reply* is received from all its neighbors before the node is allowed to send the next update message. The node is said to be in ACTIVE state during this period. The inter-neighbor synchronization used in MPATH spans only one hop, unlike algorithms that use diffusing computation that potentially span the whole network(e.g., DASM [74]).

Assume that all nodes are in PASSIVE state initially with correct distances to all other nodes and that no messages are in transit or pending to be processed. The behavior of the network where every node runs MPATH is such that when a finite sequence of link cost changes occurs in the network within a finite time interval, some or all nodes to go through a series of PASSIVE-to-ACTIVE and ACTIVE-to-PASSIVE state transitions, until eventually all nodes become PASSIVE with correct distances to all destinations.

Let a node in PASSIVE state receive an event resulting in changes in its distances to some destinations. Before the node sends an update message to report new distances, it checks if the distance D_j^i to any destination j has increased above the previously reported distance RD_j^i. If none of the distances increased, then the node remains in PASSIVE state. Otherwise, the node sets the *query* flag in the update message, sends it, and goes into ACTIVE state. When in ACTIVE state, a node cannot send any update messages or *add* neighbors to any successor set. After receiving replies from all its neighbors the node is allowed to modify the

successor sets and report any changes that may have occurred since the time it has transitioned to ACTIVE state, and if none of the distances increased beyond the reported distance, the node transitions to PASSIVE state. Otherwise, the node sends the next update message with the *query* bit set and becomes ACTIVE again, and the whole cycle repeats. If a node receives a message with the *query* bit set when in PASSIVE state, it modifies its tables and then sends back an update message with the *reply* flag set. Otherwise, if the node happens to be in ACTIVE state, it modifies the tables but because the node is not allowed to send updates when in ACTIVE state, the node sends back an empty message with no updates but the *reply* bit set. If a reply from a neighbor is pending when the link to the neighbor fails then an implicit reply is assumed, and such a reply is assumed to report an infinite distance to the destination. Because replies are given immediately to queries and replies are assumed to be given upon link failure, deadlocks due to inter-neighbor synchronization cannot occur. Eventually, all nodes become PASSIVE with correct distances to destinations, which we prove in the next section.

2.4.2 Correctness Proof of MPATH

The following properties of MPATH must be proved: (1) MPATH eventually converges with D_j^i giving the shortest distances and (2) the successor graph SG_j is loop-free at every instant and eventually converges to the shortest multipath. As mentioned earlier, PATH works essentially like PDA except that the kind of update information exchanged is different; PDA exchanges link-state while PATH exchanges distance-vectors with predecessor information. The correctness proof of PATH is identical to PDA and are reproduced here for correctness. Because PATH and PDA differ only in the way changes to main topology are reported, the proof the PATH converges is identical to that of PDA. The convergence of MPATH directly follows from the convergence of PATH because extensions to MPATH are such that update messages in

MPATH are only delayed a finite amount of time. A node generates update messages only to report changes in distances and predecessor, so after convergence MPATH generates no messages.

The following theorems show that MPATH provides instantaneous loop-freedom and correctly computes the shortest multipath.

Theorem 9 *For the algorithm MPATH executed at node i, let t_n be the time when RD_j^i is updated and reported for the n-th time. Then, the following conditions always hold.*

$$FD_j^i(t_n) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \tag{2.22}$$

$$FD_j^i(t) \leq FD_j^i(t_n) \qquad t \in [t_n, t_{n+1}) \tag{2.23}$$

Proof: From the working of MPATH in Fig. 2.12, we observe that RD_j^i is updated at line 12 when (a) the node goes from PASSIVE-to-ACTIVE because of one or more distance increases (b) the node receives the last reply and goes from ACTIVE-to-PASSIVE state (c) the node is in PASSIVE state and remains in PASSIVE state because the distance did not increase for any destination (d) the node receives the last reply but immediately goes into ACTIVE state. The reported distance RD_j^i remains unchanged during the ACTIVE phase. Because FD_j^i is updated at line 10 each time RD_j^i is updated at line 12, Eq. (2.22) follows. When the node is in ACTIVE phase, FD_j^i may also be modified by the statement on line 17, which implies Eq. (2.23). □

Theorem 10 *(Safety property) At any time t, the successor sets $S_j^i(t)$ computed by MPATH are loop-free.*

Proof: The proof is based on showing that the FD_j^i and S_j^i computed by MPATH satisfy the LFI conditions. Let t_n be the time when RD_j^i is updated and reported for the n-th

time. The proof is by induction on the interval $[t_n, t_{n+1}]$. Let the LFI condition be true up to time t_n, we show that

$$FD_j^i(t) \leq D_{ji}^k(t) \qquad t \in [t_n, t_{n+1}] \tag{2.24}$$

From Theorem 9 we have

$$FD_j^i(t_n) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \tag{2.25}$$

$$FD_j^i(t_{n+1}) \leq min\{RD_j^i(t_n), RD_j^i(t_{n+1})\} \tag{2.26}$$

$$FD_j^i(t) \leq FD_j^i(t_n) \qquad t \in [t_n, t_{n+1}) \tag{2.27}$$

Combining the above equations we get

$$FD_j^i(t) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \quad t \in [t_n, t_{n+1}] \tag{2.28}$$

Let t' be the time when message sent by i at t_n is received and processed by neighbor k. Because of the non-zero propagation delay across any link, t' is such that $t_n < t' < t_{n+1}$ and because RD_j^i is modified at t_n and remains unchanged in (t_n, t_{n+1}) we get

$$RD_j^i(t_{n-1}) \leq D_{ji}^k(t) \qquad t \in [t_n, t') \tag{2.29}$$

$$RD_j^i(t_n) \leq D_{ji}^k(t) \qquad t \in [t', t_{n+1}] \tag{2.30}$$

From Eq. (2.29) and (2.30) we get

$$min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \leq D_{ji}^k(t) \quad t \in [t_n, t_{n+1}] \tag{2.31}$$

From (2.28) and (2.31) the inductive step (2.24) follows. Because $FD_j^i(t_0) \leq D_{ji}^k(t_0)$ at initialization, from induction we have that $FD_j^i(t) \leq D_{ji}^k(t)$ for all t. Given that the successor sets are computed based on FD_j^i, it follows that the LFI conditions are always satisfied. According to the Theorem 1 this implies that the successor graph SG_j is always loop-free. □

Theorem 11 *(Liveness property) A finite time after the last change in the network, the* D_j^i *give the correct shortest distances and* $S_j^i = \{k | D_j^k < D_j^i, k \in N^i\}$.

Proof: The proof is similar to that of MPDA. □

2.4.3 Complexity Analysis of MPATH

The main difference between PATH and MPATH is that the update messages sent in MPATH are delayed a finite amount of time in order to enforce the invariants. As a result, the complexity of PATH and MPATH are essentially the same and are therefore collectively analyzed. This same analysis of PATH applies to PDA.

The *storage complexity* is the amount of table space needed at a node. Each one of the N^i neighbor tables and the main distance table has size of the order $O(|N|)$ and the main link table T^i can grow, during execution of MTU-PATH, to size at most $|N^i|$ times $O(|N|)$. The storage complexity is therefore of the order $O(|N^i||N|)$.

The *time complexity* is the time it takes for the network to converge after the last link cost change in the network. To determine time complexity we assume the computation time to be negligible as compared to the communication times. If t_n is the time when every node has the n-hop minimum tree, because every node processes and reports changes in finite time $|t_{n+1} - t_n|$ is bounded. Let $|t_{n+1} - t_n| \leq \theta$ for some finite constant θ. From theorem 6, the convergence time can be at most $|N|\theta$ and, hence, the time complexity is $O(|N|)$.

The *computation complexity* is the time taken to build the node's shortest path tree in T^i from the neighbor tables T_k^i. Updating of T^i with T_k^i information is $O(|N^i||N|)$ operation and running Dijkstra on T^i takes $O(|N^i||N|log(|N|))$. Therefore the computation complexity is $O(|N^i||N| + |N^i||N|log(|N|))$.

The *communication complexity* is the number of update messages required for propa-

gating a set of link-cost changes. The analysis for multiple link-cost changes is complex because of the sensitivity to the timing of the changes. So, we provide the analysis only for the case of single link-cost change. A node removes a link from its shortest path tree if only a shorter path using two or more links is discovered and once discovered the path is remembered. Therefore, a removed link will not be added again to the shortest path which means that a link can be included and deleted from the shortest path by a node at most one time. Because nodes report each change only once to each neighbor, an update message can travel only once on a link and therefore the number of messages sent by a node can be at most $O(|E|)$. For certain topologies and sensitively timed sequence of link cost changes the amount of communication required by PATH can be exponential. Humblet [37] provides an example that exhibits such behavior, and though PATH is different from the shortest-path algorithm presented in that paper, we note that PATH is not immune from such exponential behavior. However, we believe such scenarios require sensitively timed link-cost changes which are very unlikely to occur in practice. If necessary, a small hold-down time before sending update messages may be used to prevent such behavior.

2.5 Performance Comparison

We use simulations to compare the control overhead and convergence time of MDVA and MPDA, with DBF and topology broadcast (TOPB). The main purpose of these simulations is to give some qualitative explanation for the behavior of MDVA and MPDA. The reason for choosing DBF and TOPB is that DBF is based on vectors of distances and does not use diffusing computations, while TOPB represents an ideal upper bound on performance of the widely used routing protocols OSPF and IS-IS. MDVA achieves loop-freedom through diffusing computations that, in some cases, may span the whole network. In contrast, MPDA uses

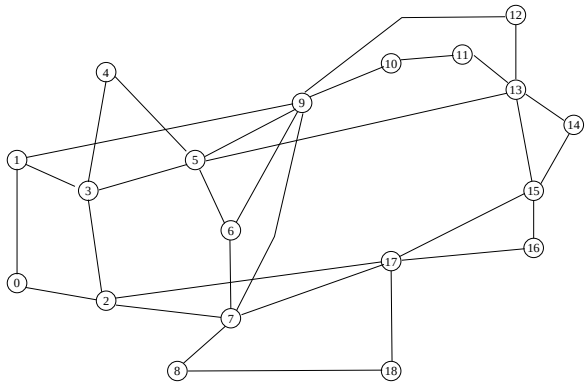


Figure 2.13: Topology used in simulations

only neighbor-to-neighbor synchronization. It is interesting to see how convergence times and control message overheads are effected by the synchronization mechanisms. A comparison of several algorithms that does not include MPDA and MDVA is given in [19].

Simulations are performed on the topology shown in Fig.(2.13). The simulator used is an event-driven real-time simulator called CPT[1]. We assume the computation time to be negligible compared to the communication times. The bandwidth and propagation delays of each link are 5MB and 100μs respectively. In backbone networks, links and nodes are highly reliable and change status much less frequently than link costs which are a function of the traffic on the link. This is particularly true in near-optimal routing (Chapter 3) is used, in which the link costs are periodically measured and reported. For this reason, in this paper we focus on comparing the algorithms in scenarios when multiple link-cost changes occur.

In each experiment, all links are initially set at unit cost and then each link cost is changed by amounts determined by the formula $\alpha k + \beta r$, The parameters of the experiment α and β are real values while k is a positive integer, and r is a uniformly distributed random value in [0, 1]. After setting the new link costs, the convergence times and message overheads

[1]We thank Nokia Wireless Routers for allowing us using the C++ Protocol Toolkit

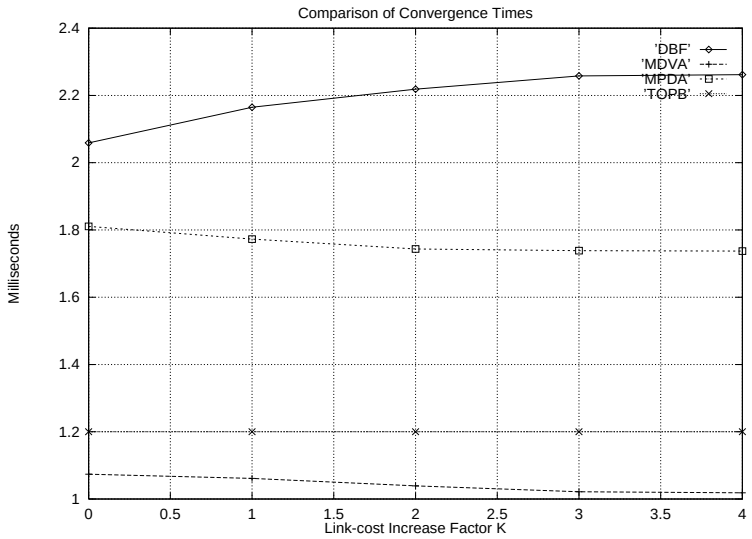


Figure 2.14: Average convergence times. $\alpha = 1$, $\beta = 5$.

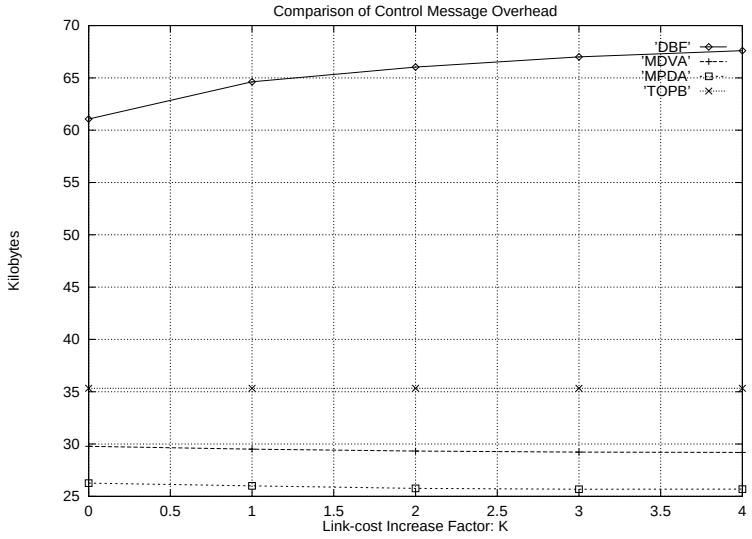


Figure 2.15: Average message overhead. $\alpha = 1$, $\beta = 5$.

are measured for each routing algorithm. For each experiment with specific α, k and β, several trials are made using different random values for r. The averages and probability distributions obtained for each metric and for each set of trials are compared.

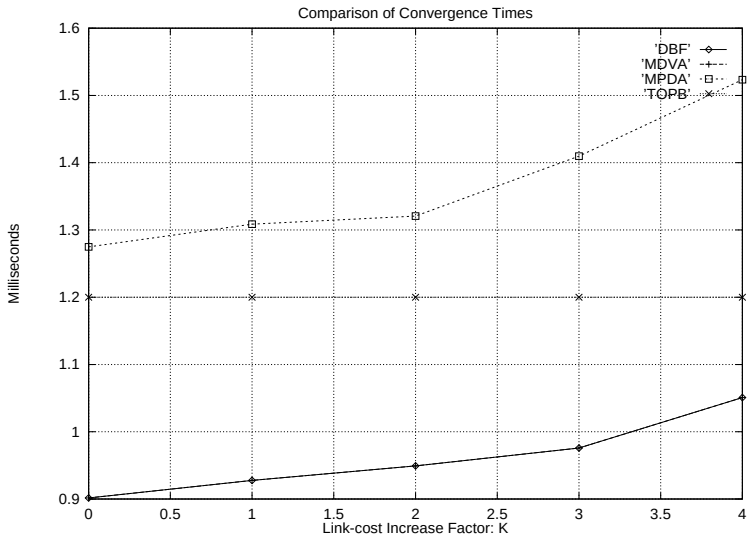


Figure 2.16: Average convergence times. $\alpha = -0.1$, $\beta = -0.4$. Note that DBF and MDVA behave identically.

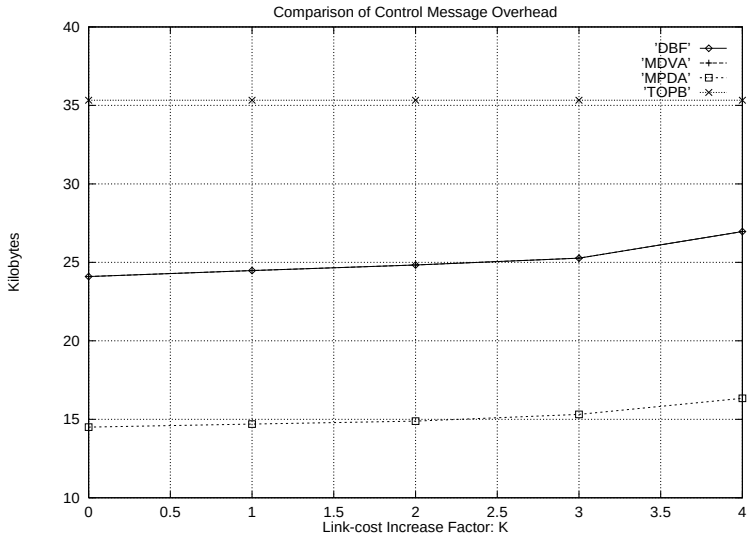


Figure 2.17: Average message overhead. $\alpha = -0.1$, $\beta = -0.4$. Note that DBF and MDVA behave identically.

Fig. 2.14 and Fig. 2.15 show the average convergence time and average message load, measured over several trials, when the links costs are increased from initial unit cost to a cost using the formula $\alpha k + \beta r$ with $\alpha = 1, \beta = 5$ and $k = 0, 1, 2, 4$. As can be observed in Fig. 2.14

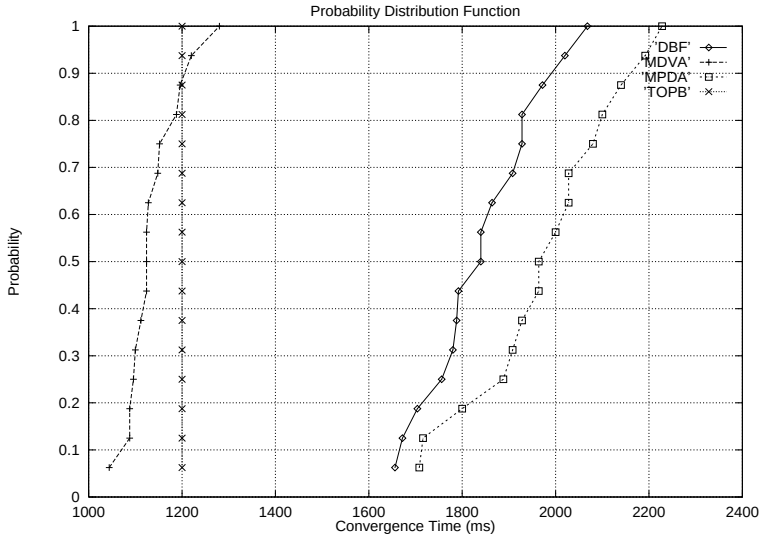


Figure 2.18: PDF of convergence times. $\alpha = 1$, $\beta = 5$, $k = 1$.

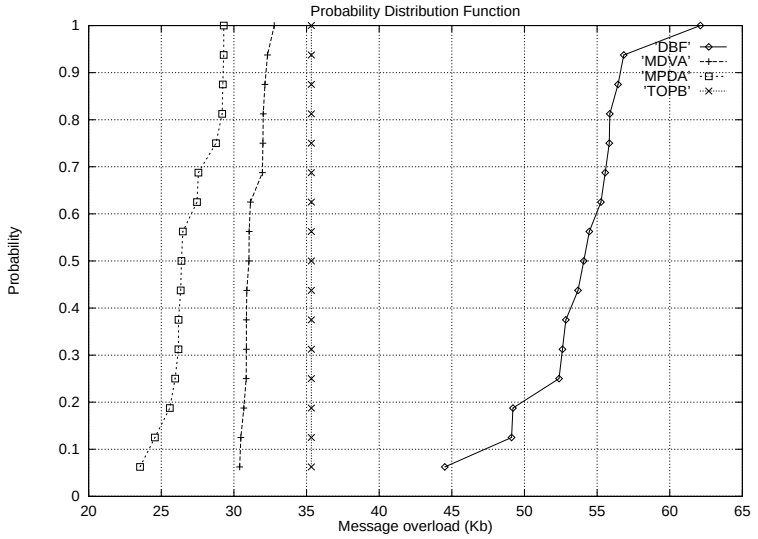


Figure 2.19: PDF of message overhead. $\alpha = 1$, $\beta = 5$, $k = 1$.

the average convergence times are best for MDVA. As can be seen in Fig. 2.15, the average message loads are also low, and only MPDA has lower message overhead. Figures 2.16 and 2.17 show the averages for convergence times and message overloads when link costs decrease.

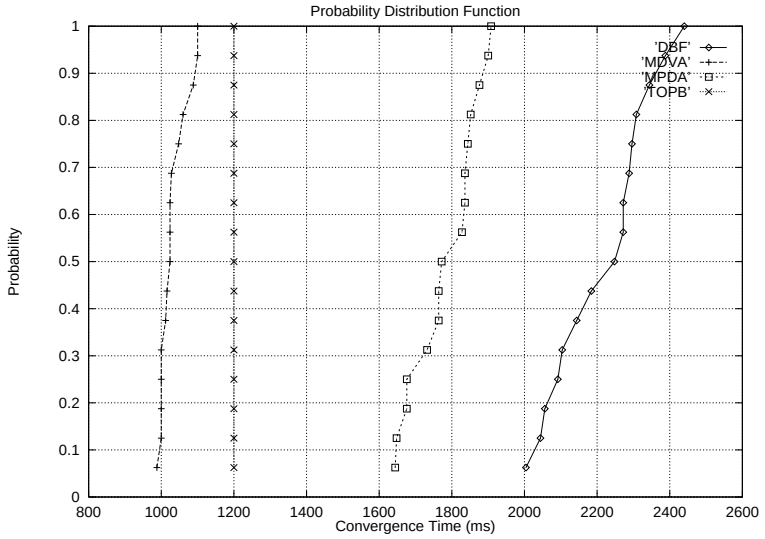


Figure 2.20: PDF of convergence times. $\alpha = 5$, $\beta = 5$, $k = 1$.

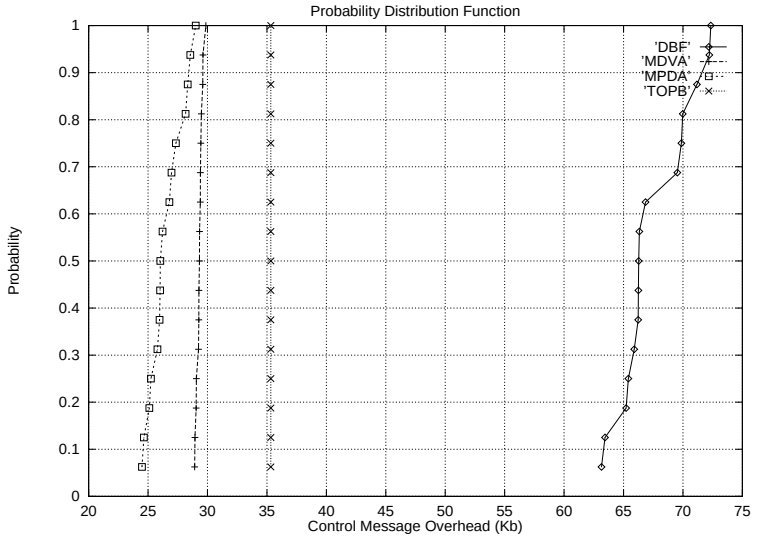


Figure 2.21: PDF of message overhead. $\alpha = 5$, $\beta = 5$, $k = 1$.

Observe that DBF and MDVA perform identically as can be seen in the figures.

Fig. 2.18 and Fig. 2.19 show the complete distribution for convergence times and message overhead for the case $k = 1, \alpha = 1, \beta = 5$. Observe that the distributions are quite

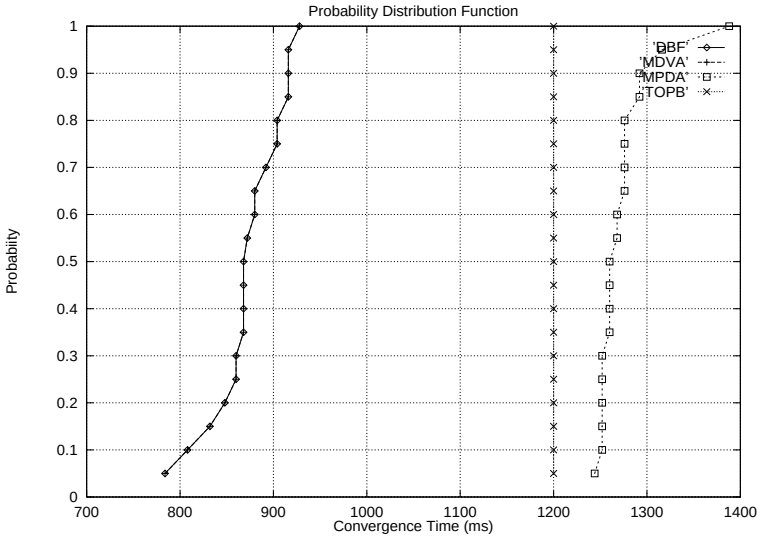


Figure 2.22: PDF of convergence times. $\alpha = 0$, $\beta = -0.4$. Note that DBF and MDVA behave identically.

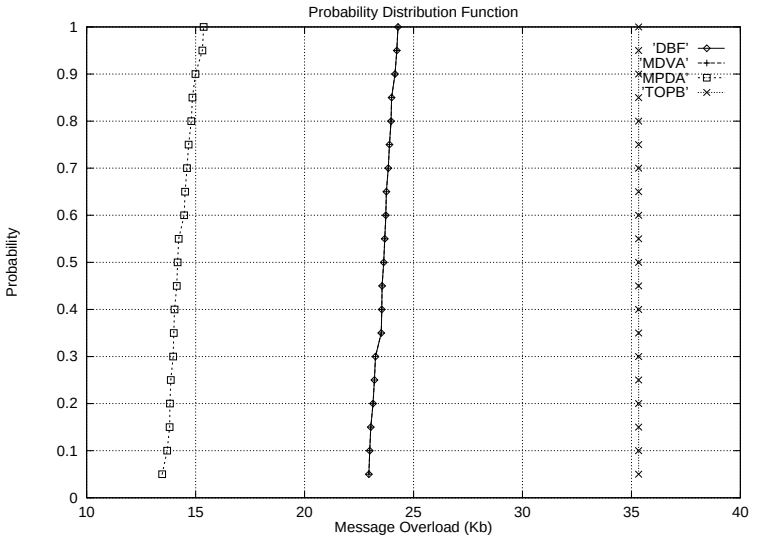


Figure 2.23: PDF of message overhead. $\alpha = 0$, $\beta = -0.4$. Note that DBF and MDVA behave identically.

uniform compared to DBF. When k is increase to 5 from 1, the convergence times and message overheads of MDVA, as shown in Fig. 2.20 and Fig. 2.21, have not changed much, but the performance of DBF has degraded considerably. This is because of the counting-to-infinity

problem, which does not occur in MDVA.

Fig. 2.22 and Fig. 2.23 show the convergence time and message overhead distribution when link costs decrease ($\alpha = 0$, $\beta = -0.4$). (Note that we make sure that link costs do not become negative.) Observe that the performance of MDVA and DBF are the same which is because MDVA essentially functions like DBF when distances to destinations decrease. From these simulations it appears that MDVA is a good choice if low convergence times are desired at the expense of high message overload while MPDA is preferable if low message overhead is desirable over convergence times.

2.6 Related Work

2.6.1 Distance-vector algorithms

RIP[35] is based on the Distributed Bellman-Ford (DBF) algorithm for computing the shortest-paths to destinations. In networks that use RIP, nodes exchange only distances to destinations and have no knowledge of the network topology, and due to lack of this information they suffer from the infamous *counting-to-infinity* problem[7]. Several techniques have been proposed to tackle this problem. DUAL[30], which is the algorithm used in EIGRP [1], uses diffusing computations [27]. In addition to DUAL, several algorithms based on distance vectors have been proposed that use diffusing computation to overcome the counting-to-infinity problem of DBF [56, 44, 38, 74]. Jaffe and Moss[38] allow nodes to participate in multiple diffusing computation of the same destination, which requires use of unbounded counters. In contrast, DUAL restricts a node to participate in only one diffusing computation at any one time for any destination. All these routing algorithms provide only one loop-free path to the destination. Zaumen and Garcia-Luna-Aceves [74] presented DASM, which is the first distance-vector algorithm that provides loop-free multipaths. DASM is shown to perform better than DUAL,

which was previously the best distance-vector algorithm. MDVA is the first distance-vector algorithm that is designed using the LFI. All the algorithms mentioned above use diffusing computations that potentially span the whole network. In contrast, MPATH uses only single-hop synchronization, i.e., a node needs to synchronize only with its neighbors. For this reason, we chose DASM as the candidate routing algorithm based on distance vectors with which to compare MPATH.

2.6.2 Link-state algorithms

In link-state algorithms, full topology information is flooded through the network. When periodic updates are made as in the case of near-optimal routing, the overhead is very high. Routing protocols based on topology-broadcast (e.g., [59, 49]) incur too much communication overhead, which forces the network administrators to partition the network into areas connected by a backbone. This makes OSPF complex in terms of router configuration required. A couple of routing algorithms have been proposed that operate using partial topology information (LVA [31],ALP [33]) to eliminate the main limitation of topology-broadcast algorithms.

In the above link-state algorithms nodes distinguish new and old link information using sequence numbers, which are not only an added overhead but also require to be reset o occasions. Instead of sequence numbers, MPDA and PATH use a novel update rule to distinguish old and new information. Like the link-state algorithms, MPDA and MPATH are free from count-to-infinity problem. In OSPF, multiple equal-cost paths are computed to each destination if they exist. However, these paths are not loop-free during network transitions, and even if short-lived, these loops may cause incorrect link-cost measurements. In contrast, MPATH maintains multiple paths that need not be of equal cost and which are loop-free at

every instant.

Several distributed shortest-path algorithms [37, 52, 32] have been proposed that use the distance and second-to-last hop to destinations as the routing information exchanged among nodes. These algorithms are often called path-finding algorithms or source-tracing algorithms. Though they exchange distances like the distance-vector algorithms, they are akin to link-state algorithms because they internally maintain path information obtained using the predecessor information; distance-vector algorithm have no knowledge of network topology. These algorithms eliminate DBF's counting to infinity problem using the path information. Some of them [32] are more efficient that any of the routing algorithms based on link-state information proposed to date. Furthermore, LPA [32] is loop-free at every instant, but provides only one path. MPATH is the first path-finding algorithm that builds multiple loop-free paths. As in LPA, the synchronization in MPATH is geared towards providing loop-free paths.

Chapter 3

Traffic Engineering based on Minimum-delay Routing

To improve performance of current IP networks, several solutions have been proposed both inside and outside of the traffic engineering framework [2]. There are several drawbacks of these solutions: (1) most of the solutions do not have any theoretical basis for optimization and those that do are not practical for implementation, (2) they introduce connection-oriented mechanisms in IP that is essentially connection-less. Instead of using ad hoc techniques to improve the poor performance of single-path routing, we approach the problem from a theoretical perspective. In this chapter we present a practical routing framework that approximate optimal routing [29]. We first derive a simple approximation to Gallager's minimum-delay routing algorithm [29] that can be implemented easily using the multipath routing algorithms developed in the previous chapter. We then show how the approximation can be implemented in current IP networks in a scalable manner without using connection-oriented techniques such as MPLS. Using simulations we compare the performance of the approximation routing frame-

work with ideal minimum-delay routing and show that the average end-to-end delays achieved to the frameworks are within a small percentage of the theoretical minimum-delays.

3.1 Minimum-delay Routing Problem Formulation

The minimum-delay routing problem (*MDRP*) was first formulated by Gallager [29], and we provide a description of it in this section. A computer network $G = (N, L)$ is made up of N routers and L links between them. Each link is bidirectional with possibly different costs in each direction.

Let $r_j^i \geq 0$ be the expected input traffic, measured in bits per second, entering the network at router i and destined for router j. Let t_j^i be the sum of r_j^i and the traffic arriving from the neighbors of i for destination j. And let routing parameter ϕ_{jk}^i be the fraction of traffic t_j^i that leaves router i over link (i,k). Assuming that the network does not lose any packets, from conservation of traffic we have

$$t_j^i = r_j^i + \sum_{k \in N^i} t_j^k \phi_{ji}^k \tag{3.1}$$

where N^i is the set of neighbors of router i.

Let f_{ik} be the expected traffic, measured in bits per second, on link (i,k). Because $t_j^i \phi_{jk}^i$ is the traffic destined for router j on link (i,k) we have the following equation to find f_{ik}.

$$f_{ik} = \sum_{j \in N} t_j^i \phi_{jk}^i \tag{3.2}$$

Note that $0 \leq f_{ik} \leq C_{ik}$, where C_{ik} is the capacity of link (i,k) in bits per second.

Property 2 *For each router i and destination j, the routing parameters ϕ_{jk}^i must satisfy the following conditions:*

1. $\phi^i_{jk} = 0$ *if* $(i,k) \notin L$ *or* $i = j$. *Clearly, if the link does not exist, there can be no traffic on it.*

2. $\phi^i_{jk} \geq 0$. *This is true, because there can be no negative amount of traffic allocated on a link.*

3. $\sum_{k \in N^i} \phi^i_{jk} = 1$. *This is a consequence of the fact that all incoming traffic must be allocated to outgoing links.*

Let D_{ik} be defined as the expected number of messages or packets per second transmitted on link (i,k) times the expected delay per message or packet, including the queuing delays at the link. We assume that messages are delayed only by the links of the network and D_{ik} depends only on flow f_{ik} through link (i,k) and link characteristics such as propagation delay and link capacity. $D_{ik}(f_{ik})$ is a continuous and convex function that tends to infinity as f_{ik} approaches C_{ik}. The total expected delay per message times the total expected number of message arrivals per second is given by

$$D_T = \sum_{(i,k) \in L} D_{ik}(f_{ik}) \tag{3.3}$$

Note that the router traffic-flow set $t = \{t^i_j\}$ and link-flow set $f = \{f_{ik}\}$ can be obtained from $r = \{r^i_j\}$ and $\phi = \{\phi^i_{jk}\}$. Therefore, D_T can be expressed as a function of r and ϕ using Eqs. (3.1) and (3.2). The minimum-delay routing problem can now be stated as follows:

MDRP: *For a given fixed topology and input traffic flow set* $r = \{r^i_j\}$, *and delay function* $D_{ik}(f_{ik})$ *for each link* (i,k), *the minimization problem consists of computing the routing parameter set* $\phi = \{\phi^i_{jk}\}$ *such that the total expected delay* D_T *is minimized.*

3.2 A Minimum Delay Routing Algorithm

Gallager [29] derived the necessary and sufficient conditions that must be satisfied to solve MDRP. These conditions are summarized in Gallager's Theorem stated below.

The partial derivatives of the total delay, D_T, of Eq.(3.3) with respect to r and ϕ play a key role in the formulation and solution of the problem; these derivatives are:

$$\frac{\partial D_T}{\partial r_j^i} = \sum_{k \in N^i} \phi_{jk}^i [D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_j^k}] \tag{3.4}$$

$$\frac{\partial D_T}{\partial \phi_{jk}^i} = t_j^i [D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_j^k}] \tag{3.5}$$

where $D'_{ik}(f_{ik}) = \partial D_{ik}(f_{ik})/\partial f_{ik}$. and is called the *marginal delay* or *incremental delay.*

Similarly, $\partial D_T / \partial r_j^i$ is called the *marginal distance* from router i to j.

Gallager's Theorem [29]: *The necessary condition for a minimum of D_T with respect to ϕ for all $i \neq j$ and $(i,k) \in L$ is*

$$\frac{\partial D_T}{\partial \phi_{jk}^i} = \lambda_{ij} \quad \phi_{jk}^i > 0 \tag{3.6}$$

$$\geq \lambda_{ij} \quad \phi_{jk}^i = 0 \tag{3.7}$$

where λ_{ij} is some positive number, and the sufficient condition to minimize D_T with respect to ϕ is for all $i \neq j$ and $(i,k) \in L$ is

$$D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_j^k} \geq \frac{\partial D_T}{\partial r_j^i} \qquad \square \tag{3.8}$$

Eq. (3.4) shows the relation between a router's marginal distance to a particular destination and the marginal distances of its neighbors to the same destination. Eqs. (3.5)-(3.8) indicate the conditions for *perfect load balancing*, i.e., when the routing parameter set ϕ gives the minimum delay. The set of neighbors through which router i forwards traffic towards j is denoted by S_j^i and is called the *successor set.* Under perfect load balancing with respect

to a particular destination, the marginal distances through neighbors in the successor set are equal to the marginal distance of the router, and the marginal distances through neighbors *not* in the successor set are higher than the marginal distance of the router.

Let D_j^i denote the *marginal distance* from i to j, i.e., $\partial D_T/\partial r_j^i$. Let the *marginal delay* $D'_{ik}(f_{ik})$ from i to k be denoted by l_k^i which is also called the cost of the link from i to k. Note that in most routing protocols, the link costs do not include queuing delays. Usually the queuing delays are significantly higher than the propagation delays.

According to Gallager's Theorem, the minimum delay routing problem now becomes one of determining, at each router i for each destination j: the routing parameters $\{\phi_{jk}^i\}$, S_j^i and D_j^i, such that the following five equations are satisfied:

$$D_j^i = \sum_{k \in N^i} \phi_{jk}^i (D_j^k + l_k^i) \tag{3.9}$$

$$S_j^i = \{k | \phi_{jk}^i > 0 \wedge k \in N^i\} \tag{3.10}$$

$$D_j^i = D_j^k + l_k^i \qquad k \in S_j^i \tag{3.11}$$

$$D_j^i < D_j^k + l_k^i \qquad k \notin S_j^i \tag{3.12}$$

This reformulation of MDRP is critical, because it is the first step in allowing us to approach the problem by looking at the next-hops and distances obtained at each router for each destination. Gallager [29] described a distributed routing algorithm for solving the above five equations. When the algorithm converges, the aggregate of the successor sets for a given destination j (S_j^i for every i) define a directed acyclic graph. In fact, in any implementation, S_j^i *must be* loop-free at every instant, because even temporary loops cause traffic to loop at some nodes and results in incorrect marginal delay computations, which in turn can prevent the algorithm from converging or obtaining minimum delays.

Gallager's distributed algorithm uses an interesting blocking technique to provide

loop-freedom at every instant [29, 55, 57]. We refer to this algorithm as OPT in the rest of the paper. Unfortunately, OPT cannot be used in real networks for several reasons. A major drawback of OPT is that a global step size η needs to be chosen and every router must use it to ensure convergence. Because η depends on the input traffic pattern, it is impossible to determine one in practice that works for all input traffic patterns and for all possible topology modifications. The routing parameters are directly computed by OPT and the multiple loop-free paths are simply implied by the routing parameters in Eq. (3.10). The computation of routing parameters is, for all practical purposes, a very slow process as it is a destination-controlled process. The destination initiates every iteration that adjusts the routing parameters at every router; furthermore, each iteration takes a time proportional to the diameter of the network and number of messages proportional to number of links. This renders the algorithm slow converging and useful only when traffic and topology are stationary for times long enough for all routers to adjust their routing parameters between changes. Also, depending on the global constant η, the destination must initiate several iterations for the parameters to converge to their final values. The number of such iterations needed for convergence tends to be large for a small η, and small for a large value of η. Unfortunately, η cannot be made arbitrarily large to reduce the number of iterations and to speed up convergence, because the algorithm may not converge at all for large values of η.

Hence, Gallager's algorithm can be viewed only as a method for obtaining lower bounds under stationary traffic, rather than as an algorithm to be used in practice. The next section shows how the theory introduced in the Gallager's method can be adapted to practical networks.

3.3 Near-optimal Routing Framework

We noted that in Gallager's algorithm the computation of the routing parameter set ϕ is slow converging and works only in the case of stationary or quasi-stationary traffic. In the Internet, traffic is hardly stationary and perfect load balancing is neither possible nor necessary. Intuitively, an approximate load balancing scheme based on some heuristic which can quickly adapt to dynamic traffic should be sufficient to minimize delays substantially.

The key idea in our approach is, in a sense, to reverse the way in which Gallager's algorithm solves MDRP. The intuition behind our approach is that establishing paths from sources to destinations takes a much longer time than shifting loads from one set of neighbors to another, simply because of the propagation and processing delays incurred along the paths. Accordingly, it makes sense to first establish multiple loop-free paths using long-term (end-to-end) delay information, and then adjust routing parameters along the predefined multiple paths using short-term (local) delay information.

This new approach allows us to attempt to use distributed algorithms to compute multiple loop-free paths from source to destination that, hopefully, are as fast as today's single-path routing algorithms, and local heuristics that can respond quickly to temporary traffic bursts using local short-term metrics alone. Therefore, we map Eqs. (3.9)-(3.12) derived in Gallager's method into the following three equations:

$$D_j^i = min\{D_j^k + l_k^i | k \in N^i\} \tag{3.13}$$

$$S_j^i = \{k | D_j^k < D_j^i \wedge k \in N^i\} \tag{3.14}$$

$$\phi_{jk}^i = \Psi(k, A_j^i, B_j^i) \qquad k \in N^i \tag{3.15}$$

where $A_j^i = \{D_j^p + l_p^i | p \in N^i\}$ and $B_j^i = \{\phi_{jp}^i | p \in N^i\}$.

These equations simply state that, for an algorithm to approximate minimum-delay

routing, it must establish loop-free paths and use a function Ψ to allocate flows over those paths. We observe that Eq. (3.13) is the well-known Bellman-Ford (BF) equation for computing the shortest paths, and Eq. (3.14) is the successor set consisting of the neighbors that are closer to the destination than the router itself. Note that the paths implied by the neighbors in the successor set of a router need not be of the same length. The function Ψ in Eq. (3.15) is a heuristic function that determines the routing parameters. Because changing the routing parameters effects the marginal delay of the links (hence link-costs), we use regular updates of the link costs.

The main problem with attempting to solve MDRP using Eqs. (3.13) to (3.15) directly is that these equations assume that routing information is consistent throughout the network. In practice, a node (router) must choose its distance and successor set using routing information obtained through its neighbors, and this information may be outdated. At any time t, for a particular destination j, the successor sets of all nodes define a *routing graph* $SG_j(t) = \{(m,n)|n \in S_j^m(t),\ m \in N\}$. In single-path routing, $S_j^i(t)$ has at most one neighbor: the neighbor that is on the shortest path to destination j. Accordingly, $SG_j(t)$ for single-path routing is a sink-tree rooted at j if loops are never created. The routing graph $SG_j(t)$ in our case should be a directed acyclic graph in order for minimum delays to be approached.

The blocking technique used in Gallager's algorithm ensures instantaneous loop-freedom. Likewise, to provide loop-free paths even when the network is in transient state within the context of our framework, additional constraints must be imposed on the choice of successors at each router, which essentially must preclude the use of neighbors that *may* lead to looping.

With the result of LFI Theorem 1, Eq. (3.14) can be approximated with the LFI conditions to render a routing approach that does not require routing information to be globally consistent, at the expense of rendering delays that may be longer than optimal. Accordingly,

our framework for near-optimum-delay routing lies in finding the solution to the following equations using a distributed algorithm:

$$D_j^i = min\{D_j^k + l_k^i | k \in N^i\} \tag{3.16}$$

$$FD_j^i \leq D_{ji}^k \qquad k \in N^i \tag{3.17}$$

$$S_j^i = \{\ k \mid D_{jk}^i < FD_j^i \wedge k \in N^i\} \tag{3.18}$$

$$\phi_{jk}^i = \Psi(k, \{D_j^p + l_p^i | p \in N^i\}, \{\phi_{jp}^i | p \in N^i\})\ k \in N^i \tag{3.19}$$

The critical component of the implementation is the loop-free multipath routing algorithm for we described three algorithms MDVA, MPDA and MPATH. Any one of them can be used. The next section describes the implementation of the framework.

3.4 Implementation of Near-optimal Routing Framework

3.4.1 Distributing Traffic over Multiple Paths

In general, the function Ψ can be any function that satisfies Property 2, but our objective is to obtain a function Ψ that performs load balancing that is as close as possible to perfect load balancing (Eqs.(3.11)-(3.12)).

The function Ψ should also be suitable for use in dynamic networks, where the flows over links are continuously changing, causing continuous link-cost changes. To respond to these changes, queuing delays at the links must be measured periodically and routing paths must be recomputed. However, re-computing paths frequently consumes excessive bandwidth and may also cause oscillations. Therefore, routing-path changes should only be done at sufficiently long intervals. Unfortunately, a network cannot be responsive to short-term traffic bursts if only long-term updates are performed. For this reason, we use link costs measured over two

00. **procedure** IH
01. **begin**
02. $\forall k \notin S_j^i, \phi_{jk}^i \leftarrow 0$;
03. **if** $(|S_j^i| = 1)$ **then**
04. $\forall k \in S_j^i, \phi_{jk}^i \leftarrow 1$;
05. **endif**
06. **if** $(|S_j^i| > 1)$ **then**
07. $\phi_{jk}^i \leftarrow \frac{1 - \frac{D_{jk}^i + l_k^i}{\sum_{m \in S_j^i} (D_{jm}^i + l_m^i)}}{(|S_j^i| - 1)}, \quad \forall k \in S_j^i$;
08. **endif**
09. **end** IH

Figure 3.1: Heuristic for initial load assignment.

different intervals; link costs measured over short intervals of length T_s are used for routing-parameter computation and link costs measured over longer intervals of length T_l are used for routing-path computation [40]. In general, T_l must be several times longer than T_s. Long-term updates are designed to handle long-term traffic changes and are used by the routing protocol to update the successor sets at each router, so that the new routing paths are the shortest paths under the new traffic conditions. The short-term updates made every T_s seconds are designed to handle short-term traffic fluctuations that occur between long-term routing path updates and are used to compute the routing parameters ϕ_{jk}^i in Eq. (3.15) locally at each router. Accordingly, our traffic distribution heuristics assume a constant successor set and successor graph.

When S_j^i is computed for the first time or recomputed again due to long-term route changes, traffic should be freshly distributed. In this case, the allocation heuristic function Ψ is a function of only the marginal distances through the successor set. That is, Eq. (3.15) reduces to the form $\{\phi_{jk}^i\} = \Psi(k, \{D_j^p + l_p^i | p \in N^i\})$. When a new successor set S_j^i is computed, algorithm IH in Fig. 3.1 is first used to distribute traffic over the successor set [40]. Note that $\{\phi_{jk}^i\}$, computed in IH, satisfy Property 1. Furthermore, when more than one successor is present, if $D_{jp}^i + l_p^i > D_{jq}^i + l_q^i$ for successors p and q, then $\phi_{jp}^i < \phi_{jq}^i$. The heuristic makes sense

00. **procedure** AH
01. **begin**
02. $D^{ij}_{min} \leftarrow min\{D^i_{jk} + l^i_k \mid k \in S^i_j\}$;
03. **let** $D^{ij}_{min} = (D^i_{jk_0} + l^i_{k_0})$;
04. // That is, k_0 be the neighbor
05. that offers this minimum)
06. **foreach** $k \in S^i_j$ **do**
07. $a^i_{jk} \leftarrow D^i_{jk} + l^i_k - D^{ij}_{min}$;
08. **done**
09. $\Delta \leftarrow \frac{1}{2} min\{\frac{\phi^i_{jk}}{a^i_{jk}} \mid k \in S^i_j \wedge a^i_{jk} \neq 0\}$;
10. **foreach** $k \neq k_0 \wedge k \in S^i_j$ **do**
11. $\phi^i_{jk} \leftarrow \phi^i_{jk} - \Delta \times a^i_{jk}$;
12. **done**
13. **for** $k = k_0$ **do**
14. $\phi^i_{jk} \leftarrow \phi^i_{jk} + \sum_{q \in S^i_j} \Delta \times a^i_{jq}$;
15. **done**
16. **end** AH

Figure 3.2: Heuristic for incremental load adjustment.

because the greater the marginal delay through a particular neighbor becomes, the smaller the fraction of traffic that is forwarded to that neighbor.

After the first flow assignment is made over a newly computed successor set using algorithm IH, a different flow allocation heuristic algorithm AH shown in Fig. 3.2 is used to adjust the routing parameters every T_s seconds until the successor set changes again. The heuristic function Ψ computed in AH is incremental and, unlike IH, is a function of current flow allocation on the successor sets and the marginal distances through the successors. AH also preserves Property 2 at every instant. In AH traffic is incrementally moved from the links with large marginal delays to links with the least marginal delay. The amount of traffic moved away from a link is proportional to how large the marginal delay of the link is compared to the best successor link. The heuristic tends to distribute traffic in such a way that Eqs. (3.11)-(3.12) hold true. This is important, because the initial distribution obtained by IH is far from being balanced. The computation complexity of the heuristic allocation algorithms is $O(N^i)$. Because the heuristics are run for each active destination, the whole load-balancing activity is

$O(N)$.

Unlike η in Gallager's algorithm, T_l and T_s are local constants that are set independently at each router. Convergence of our algorithm does not critically depend on these constants like optimal routing does on η. Also, T_l and T_s need not be static constants and can be made to vary according to congestion at the router. The value of T_l, however, should be such that it is sufficiently longer than the time it takes for computing the shortest paths. The long-term update periods should be phased randomly at each router, because of the problems that would result due to synchronization of updates [9].

3.4.2 Computing Link Costs

As mentioned earlier, the cost of a link is the marginal delay over the link $D'(f_{ik})$. If the links are assumed to behave like M/M/1 queues, then the marginal delay $D'(f_{ik})$ can be obtained in a closed form expression by differentiating the following equation [39].

$$D_{ik}(f_{ik}) = \frac{f_{ik}}{(C_{ik} - f_{ik})} + \tau_{ik} f_{ik} \tag{3.20}$$

where f_{ik} is the flow through the link (i, k), and C_{ik} and τ_{ik} are the capacity and propagation delay of the link. Because the M/M/1 assumption does not hold in practice in the presence of very bursty traffic, and because Eq. (3.20) becomes unstable when f_{ik} approaches C_{ik}, an on-line estimation of the marginal delays is desirable.

There are several techniques for computing marginal delays that are currently available (e.g., [55, 54, 11]). For the purposes of simulations, we borrow a technique introduced by Cassandras, Abidi and Towsley [11] for on-line estimation of the marginal delay $D'(f_{ik})$. The technique uses perturbation analysis (PA) for the on-line estimation and is shown to perform better than the M/M/1 estimation. In addition, the PA estimation does not require a priori

knowledge of the link capacities. This is very significant, because the capacity available to best-effort traffic in real networks varies according to the capacity allocated to other types of traffic, such as real-time traffic. We must emphasize that our approach does not depend on which specific technique is used for marginal-delay estimation, although some methods may be better than others. The convergence or stability of our routing algorithm does not depend on the specific technique used for marginal-delay estimation.

3.5 Performance of Near-optimal Routing framework

The simulations discussed in this section illustrate the effectiveness of our near-optimal framework, and demonstrate the significant improvements achieved by our approach over single-path routing in static and dynamic environments. The delays obtained by optimal routing, single-path routing and our approximation scheme are compared under identical topological and traffic environments. The results show that the average delays achieved via our approximation scheme are comparable (within a small percentage difference rather than several times difference) to the optimal routing under quasi-static environment and the same are significantly better than single-path routing in a dynamic environment.

For optimal routing, we implemented the algorithm described by Gallager [29], and label it with 'OPT'. The plots of our approximation scheme are labeled with 'MP'. To obtain representative delays for single-path routing algorithms, we opted to restrict our multipath routing algorithm to use only the best successor for packet forwarding, instead of simulating any specific shortest-path algorithm. Because of the instantaneous loop-freedom property that MPDA exhibits, the shortest-path delays obtained this way are better than or similar to the delays obtained with either EIGRP [1], which is based on DUAL and requires much more inter-nodal synchronization than our scheme, rendering longer delays, and RIP [35] or OSPF [45],

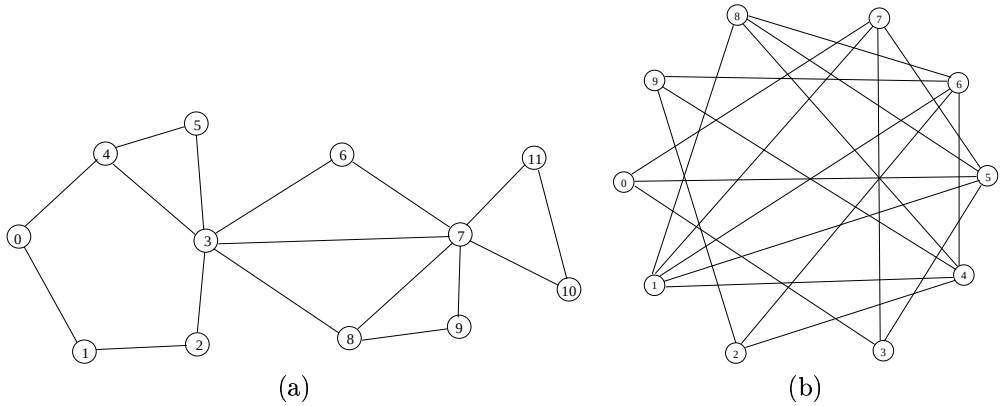


Figure 3.3: Topologies used in simulations

which do not prevent temporary loops. We use the label 'SP' for single-path routing in the graphs.

We performed simulations on the topologies shown in Fig. 3.3. CAIRN (www.cairn.net) shown in Fig. 3.3(a) is a real network and NET1 in Fig.3.3(b) is a contrived network. We are only interested in the connectivity of CAIRN, and its topology as used differs from the real network in the capacities and propagation delays assumed in the simulation experiments. We restricted the link capacities to a maximum of 10Mbs, so that it becomes easy to sufficiently load the networks. NET1 has a connectivity that is high enough to ensure the existence of multiple paths, and small enough to prevent a large number of one-hop paths. The diameter of NET1 is four and the nodes have degrees between 3 and 5. In each network we setup flows between several source-destination pairs and measure the average delays of each flow.

The flows have bandwidths in the range 0.2-1.0 Mbs. For simplicity, we used a stable topology (links or nodes do not fail) in all the simulations. In the presence of link failures, MP can only perform better than SP, because of availability of alternate paths. Furthermore, OPT is not fast enough to respond to drastic topology changes. Because MP is parameterized by

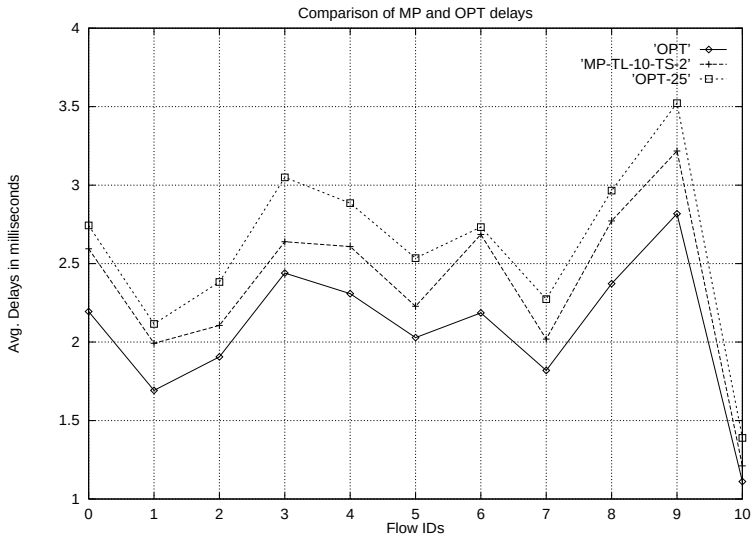


Figure 3.4: Delays of OPT and MP in CAIRN.

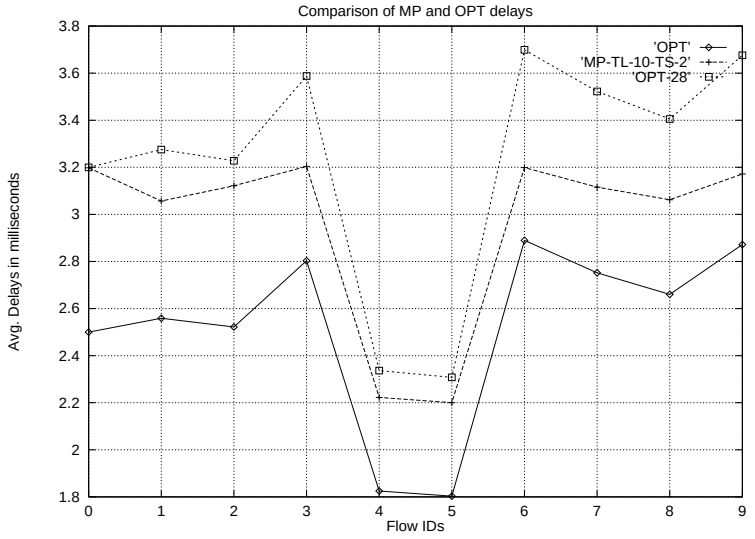


Figure 3.5: Delays of OPT and MP in NET1.

the T_l and T_s update intervals, its delay plots are represented by MP-TL-xx-TS-yy, where xx is the T_l update interval and yy is the T_s update interval measured in seconds. Similarly, the

delays of shortest-path routing are represented by SP-TL-xx, where xx is the T_l update period.

3.5.1 Performance under Stationary Traffic

Fig. 3.4 shows the average delays of flows in CAIRN for OPT and MP routing. The flow IDs are plotted on the x-axis and average delays of the flows are plotted on the y-axis. Plot OPT-25 represents the 25% 'envelope', that is, the delays of OPT are increased by 25% to obtain the OPT-25 plot. As can be seen, the average delays of flows under MP routing are within the OPT-25 envelope. Similarly, in Fig. 3.5, the delays obtained using MP routing for NET1 are within 28% envelopes of delays obtained using OPT routing. We say delays of MP are 'comparable' to OPT if the delays of MP are within a small percent of those of OPT.

Fig. 3.6 compares the average delays of MP and SP for CAIRN. We observe that the delays of SP for some flows are two to four times those of MP. In Fig. 3.7, for NET1, MP routing performs even better; average delays of SP are as much as five to six times those of MP routing which is due to higher connectivity available in NET1. Also observe that, because of load-balancing used in MP, the plots of MP are less jagged than those of SP. MP routing performs much better than SP under high-connectivity and high-load environments. When connectivity is low or network load is light, MP routing cannot offer any advantage over SP.

3.5.2 Effect of Tuning Parameters T_l and T_s

The performance of MP depends on the update intervals T_l and T_s. The setting of T_l and T_s, however, is simple. They are local and can be set independently at each node without affecting convergence, unlike the global constant η which is critical for convergence of OPT. For CAIRN, Fig. 3.8 show the effect of increasing T_l when T_s and the input traffic is fixed. Observe that when T_l is increased from 10 to 20 seconds, the delays in SP have more than doubled, while the delays of MP remain relatively unchanged. This effect indicates that T_l can be made

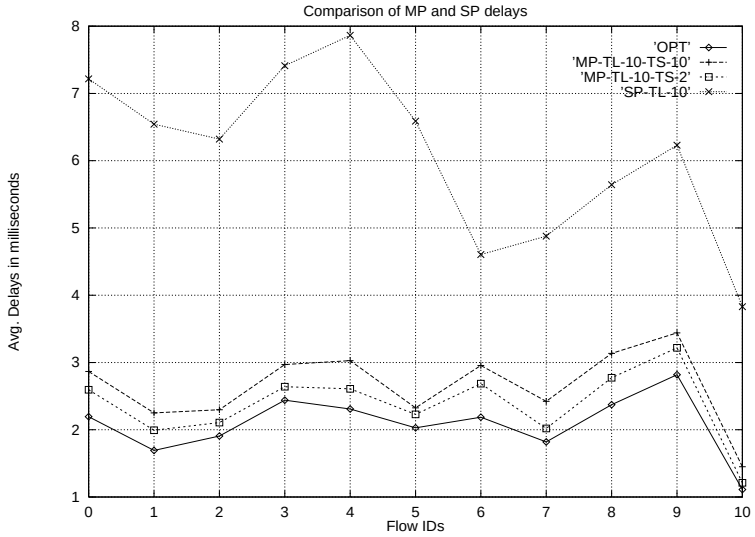


Figure 3.6: Delays of MP and SP in CAIRN.

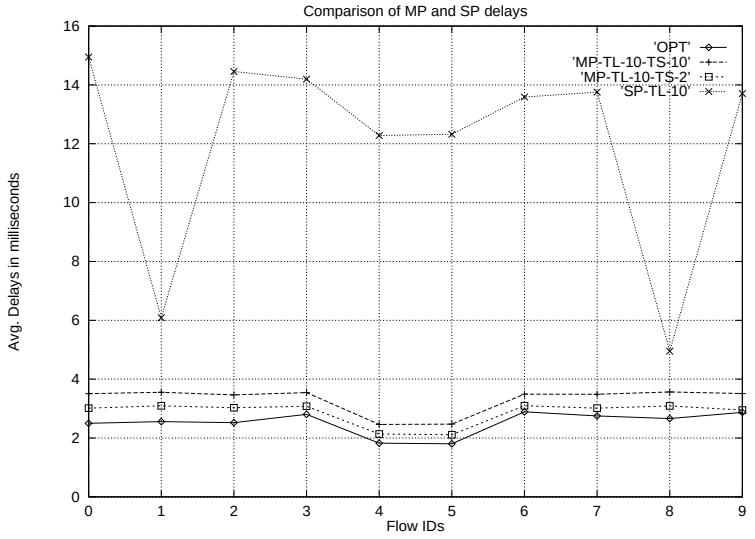


Figure 3.7: Delays of MP and SP in NET1.

longer in MP without significantly effecting performance. This is significant, because sending frequent update messages consume bandwidth and can also cause oscillations under high loads.

Similarly, for NET1, delays for SP increased significantly while there is negligible change in delays of MP as can be observed in Fig. 3.9, respectively. Our new routing framework provides the means for a trade-off between update messages and local load-balancing.

At T_s intervals, the load-balancing heuristics are executed, which are strictly local computations and require no communication. Therefore, T_s can be set according to the processing power available at the router. T_l can be made from a few times to orders of magnitude greater than T_s. In the simplest case, T_s can be set to the same value of T_l and *still* gain significant performance as shown in Figs. 3.6 and 3.7. In the figures, we observe that MP-TL-10-TS-10 is much closer to OPT than SP-TL-10. Just the long-term routes with load-balancing, without short-term routing parameter updates, seem to give significant gains; the major gains here are due to the mere presence of multiple successors and load-balancing. Our experience from simulations indicates that a T_l that is only a few times of longer than T_s suffices to gain significant benefits. This is great news, because it means that fine tuning of T_l and T_s is not important for our approach to be efficient.

3.5.3 Performance under Dynamic Traffic

It was stated earlier that OPT has very poor response to traffic fluctuations. This becomes evident in Fig. 3.10(a), which shows a typical response in NET1 when the flow rate is a step function (i.e.., the flow rate is increased from 0 to a finite amount at time 0). The dampened response of the network using MP indicates the fast responsiveness of MP, making it suitable for dynamic environments. Because OPT cannot respond fast enough to traffic fluctuations, it is impossible to find the optimal delays for dynamic traffic. However, we can find a reasonable lower bound if the input traffic pattern is predictable like the pattern shown in Fig 3.10(b), which shows only one cycle of the input pattern. To obtain a lower bound

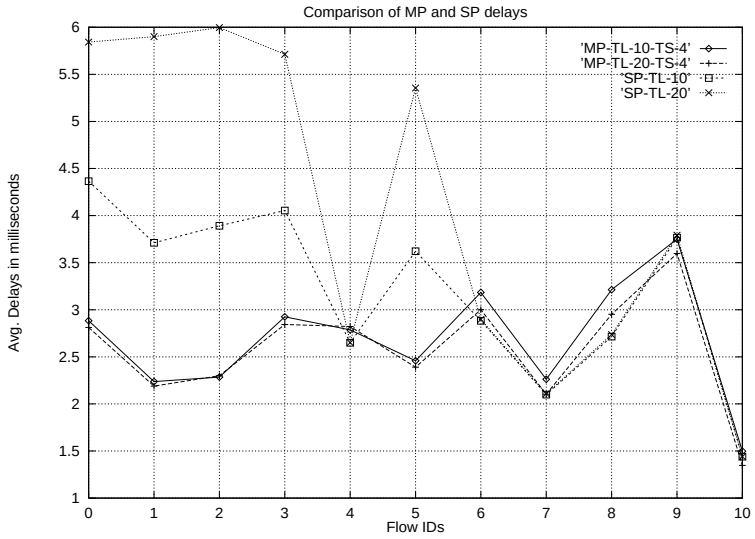


Figure 3.8: Delays when T_s is kept constant and T_l is increased in CAIRN.

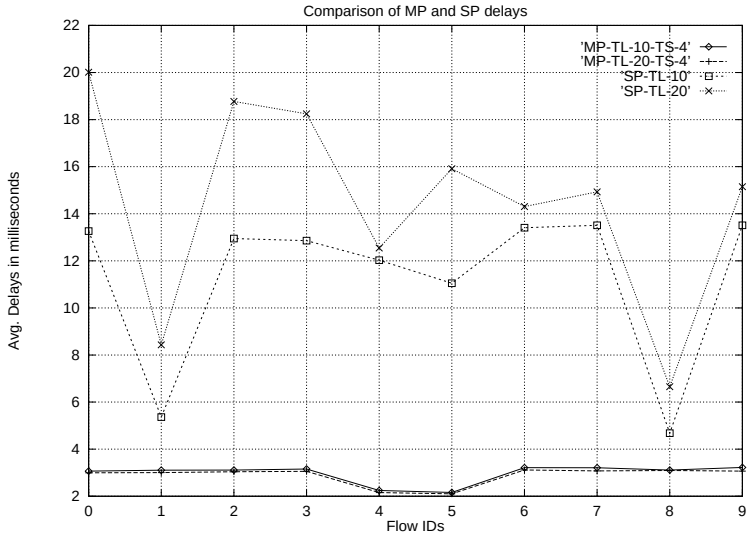
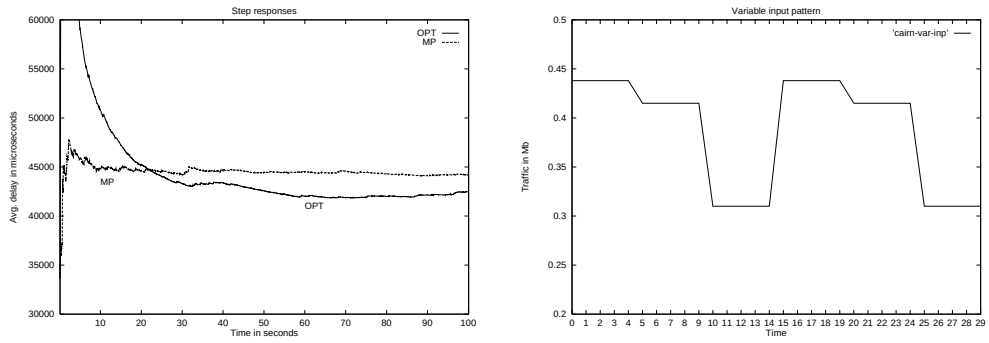


Figure 3.9: Delays when T_s is kept constant and T_l is increased in NET1.

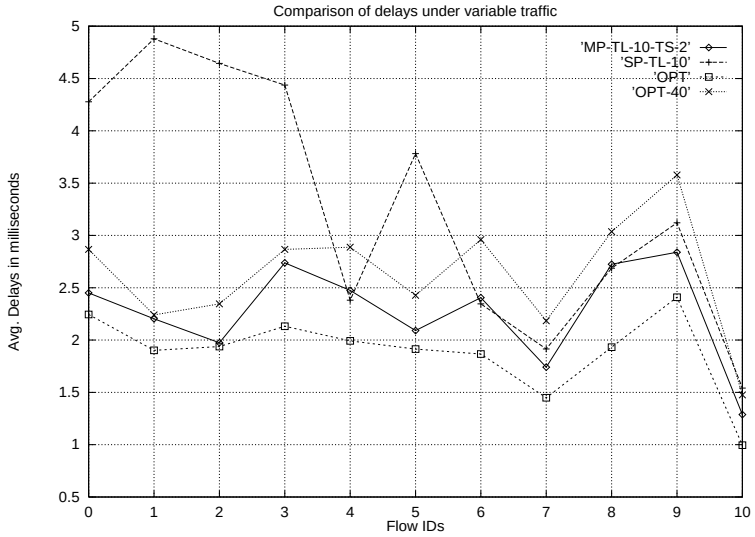


(a) (b)

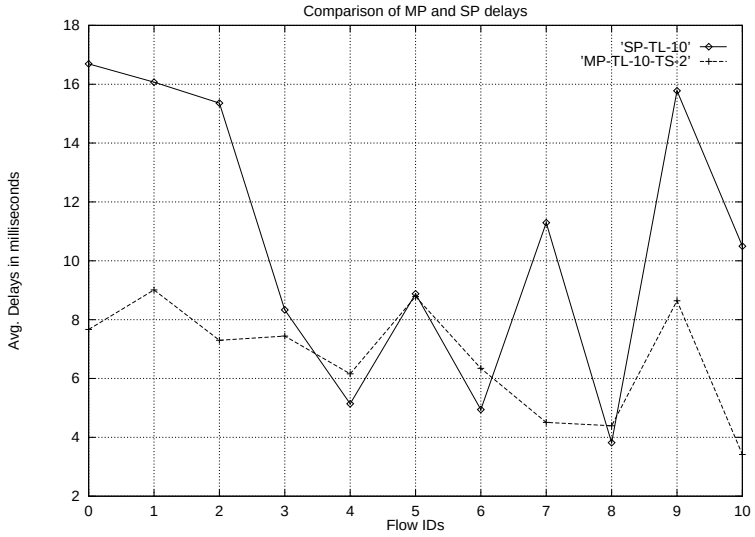
Figure 3.10: (a) Step response in NET1 using OPT and MP routing. (b) Variable input traffic pattern

for this traffic pattern that represents 'ideal' OPT (the one that has instantaneous response) we first obtain the lower bound for each interval during which traffic is steady by running a separate off-line simulation with traffic rate that corresponds to that interval, and combine the results to obtain the *lower bound*. It is with this lower bound that we compare delays of MP. Fig. 3.11 shows the average delays of the flows for OPT, MP and SP routing. The results indicate that delays of MP routing are again in the comparable range of delays of an 'ideal' optimal-routing algorithm.

Ultimately, MP will be used in real networks where traffic is bursty at any time-scale; therefore, it is important to see how MP performs in that environment. We extracted 10 flows from the Internet traffic traces obtained from LBL [24] and used them as input for the 10 flows in the CAIRN. Fig. 3.11(b) shows the delays for SP and MP. We do not perform this simulation with OPT because Internet traffic is too bursty for OPT to converge. Observe that, except for flows 4, 6 and 8, delays of MP are much better than those of SP. The reason SP delays of these flows are better than those of MP is because of uneven distribution of load in the network and low loads in some sections of the network — in low-load environments SP can perform slightly better than MP. This can be easily rectified by modifying IH to use a small threshold cost for



(a) Delays under variable traffic in CAIRN.



(b)Delays under Internet traffic in CAIRN.

Figure 3.11:

the best link, the crossing of which actually triggers the load-balancing scheme.

3.6 Minimizing Out-of-Order Packet Delivery

The near-optimal framework describe so far has the shortcoming of delivering packets out-of-order even when topology and routing parameters is stable, because the weighted round-robin (WRR) schedules packets on per-packet basis rather than on flow basis. To mitigate this, we make enhancements to the basic framework so that it deliver packets of a flow (based on addresses and ports) as best as possible. The route computation and routing parameters that routers use can be determined on-line as described in the previous section, which is the preferred way, or off-line using a central algorithm. If the off-line approach is used, then the routing parameters obtained are downloaded into the routers using a signaling protocol. The packet forwarding mechanism of the routing architecture then ensures that traffic is forwarded according to those parameters. In either case, the WRR at router i handles at most N^i inputs for each destination. To simplify description, we will assume the routing parameters are already deposited in the routing tables ready for use.

Achieving a precise packet forwarding that reflects routing parameters is non-trivial for two reasons: non-zero packet sizes and the requirement that some traffic cannot be delivered out-of-order. The WRR distributor is inadequate to handle this situation and a more sophisticated mechanism is needed. The rest of the section describes this mechanism. We assume that there are two types of traffic: one tolerates out-of-order packet delivery (e.g.,UDP) and the other does not (e.g.,TCP). For convenience we will denote by UDP, the traffic that does not require in-order delivery, and by TCP, the traffic that requires in-order delivery. Granularity of allocation for TCP traffic is at the flow level, while for UDP traffic it is at the packet level. In the presence of only UDP traffic, fairly accurate traffic distribution can be easily achieved using WRR. To deliver TCP traffic in-order, a straightforward method that is often suggested is using a hashing function on the packet's source-destination address to

determine the next-hop [64, 66]. In OSPF-OMP [66], only the source-destination pair is used in the CRC16 hashing function, which gives only a coarse distribution. By using TCP port numbers in addition to source-destination addresses in the hash function, a finer distribution can be achieved. For example, when we XOR-hashed the 1.78 million packets in the trace obtained from *http://ita.ee.lbl.gov/html/contrib/LBL-TCP-3.html* into a 32-entry hash space, using source-destination addresses in one experiment and source-destination addresses with port numbers in another, we found that the standard deviations were 61376.9 and 22558, respectively. Though these experiments are preliminary, the point we make is that we stand to benefit from using TCP port information. However, hash on port numbers can be expensive to implement in OSPF-OMP because unpacking of IP packets is required at each hop. Also in OSPF-OMP, both UDP and TCP traffic are handled identically. We show that, greater conformity between routing parameters and actual traffic distribution can achieved by treating UDP and TCP traffic differently and accounting for different packet sizes. To summarize, our approach decouples hash computation from source-destination addresses and uses a hybrid packet forwarding mechanism to improve granularity in traffic distribution.

3.6.1 Forwarding Datagram traffic

Before proceeding to describe TCP traffic and hybrid packet-forwarding, we will show that, if only UDP traffic is present and L is the maximum packet size, traffic can be forwarded such that atmost L amount of more traffic is forwarded than the traffic that would be allowed by the routing parameters. This limitation imposed by the packet-size is fundamental as packet transmission is non-preemptive.

Fig. 3.12 above describes the procedure for forwarding when only UDP traffic is present. It can be viewed as a generalization of single-path routing; in single-path routing,

all packets are forwarded to a single next-hop, whereas here, packets are forwarded to each of the several next-hops in proportions specified by the routing parameters. This procedure is extended later to handle both UDP and TCP traffic. In Fig.3.12, W_j^i is the total traffic that node i receives and forwards for j, and W_{jk}^i is the portion of that traffic that is forwarded to neighbor k. When a packet for destination j is received, the node first checks which neighbor k has a deficit and forwards it to that neighbor, after which W_j^i and W_{jk}^i are updated accordingly. The procedure allows at most L bits in excess than what the routing parameters allow. This is the finest distribution one can obtain without breaking packets into smaller units. The proof is given below.

For a given routing parameters we have to show that the forwarding algorithm in Fig. 3.12 does not forward more than L bits to any of the next-hop, and hence the traffic distribution is fairly accurate for UDP traffic even on a small scale. And the maximum deficit is $(k-1)L$. For simplicity, we slightly modify the notation. Assume an input stream with rate ρ is divided into k streams according the routing parameters. Let W be the amount of traffic that arrived so far and W_k, the amount of traffic that is forwarded to stream k and ϕ_k be the corresponding routing parameter. Also W^p denotes the value of W when the p^{th} packet arrives. $A(t,\tau)$ is the amount of input traffic that arrives in the interval $[t,\tau]$ and let the amount of traffic that stream k receives in the same interval be $A(t,\tau)$. Note that at time t, W will be equal $A[0,t]$.

Theorem 12 *In the algorithm in Fig. 3.12 (a) for each output stream k,* $(K-1)L \leq A_k(t,\tau) \leq L + \phi_k\rho(t-\tau)$.

Proof: If a packet could not be scheduled by the algorithm, then $\forall k, W_k > \phi_k W$. This implies $\sum_k W_k > W$ which is impossible. This proves that every packet is successfully scheduled by the algorithm.

```
01. procedure datagram-forwarding(P)
02. {Executed at i on arrival of the datagram packet P for j.}
03. begin
04.       Let W^i_jk ≤ φ^i_jk W^i_j, for some k ∈ S^i_j;
05.       Forward P to neighbor k;
06.       W^i_jk ← W^i_jk + sizeof(P);
07.       W^i_j ← W^i_j + sizeof(P);
08. end
```

Figure 3.12: Forwarding UDP traffic according to routing parameters.

We first show $W_k \leq L + \phi_k W$. Let it be true upto processing of $p-1$ packets. Then, for all i, $W_k^{p-1} \leq L + \phi_k W^{p-1}$. Let the new packet p be assigned to queue k, we have $W_k^p \leq W_k^{p-1} + L$. This implies $W_k^p \leq L + \phi_k W^{p-1}$ because $W_k^{p-1} \leq \phi_k W^{p-1}$. Substituting $W^p = W^{p-1} + L$, we get $W_k^p \leq L + \phi_k W^p - L\phi_k$. Therefore $W_k^p \leq L + \phi_k W^p$. Because at initialization $W_k = W = 0$, from induction it follows $W_k \leq L + \phi_k W$. Because $W_k = A_k(0,t)$ and $W = A(0,t) \leq \rho t$, we have $A_k(0,t) \leq L + \phi_k \rho t$. It follows that $A_k(0,t) \leq \delta_1 + \phi_k \rho t$ and $A_k(0,\tau) \leq \delta_2 + \phi_k \rho \tau$ for $\delta_1, \delta_2 \in [0, L]$. Therefore, $A_k(t,\tau) \leq (\delta_1 - \delta_2) + \phi_k \rho (t - \tau)$ and $(\delta_1 - \delta_2) \leq L$. That $(K-1)L \leq A_k(t,\tau)$ directly follows. □

3.6.2 Forwarding TCP traffic

We assume there are sufficiently large number of flows passing through a router and the number of flows is several order of magnitude greater than the number of next-hop choices. The duration of TCP connections, the average rates of TCP connections and the packet sizes are all uniformly distributed so that bandwidth of a group of TCP connections is proportional to number of connections in the group. In backbone networks where there are large number of flows, we believe this assumption is fairly reasonable. When the number of flows is low, however, the distribution is relatively imprecise. This, however, should be acceptable because when network load is low delays due to congestion are not as serious a problem. We will

proceed with these assumptions, and present a procedure for packet forwarding of TCP traffic. In section 3.6.4, we will give some experimental results to validate some of these assumptions. It should be noted that, under heavy load conditions, the performance of the routing scheme in the worst case only drops to that of single-path routing.

As mentioned earlier, we use an architecture similar to the Diffserv model with the obvious advantage that it can be incorporated along with other differentiated services. A bit in the TOS or DSFIELD field (codepoint) of the packet is used as an in-order-bit to indicate whether the packet is of TCP or UDP type. For each TCP connection, identified by the source-destination address and source-destination port numbers, the ingress/edge router generates a random key and stores it in a connection table. When a packet of that connection arrives at the ingress router, the key is inserted and the in-order-bit is set before forwarding it in the domain network. Within the core network the key is used to determine the next-hop at each router. Thus the computation of a hash is decoupled from the source and destination addresses. We suggest using the 13-bit fragment offset field of IP for storing the hash key. The 13-bit fragment offset field is set with the hash value only for the TCP packets and only when offset is zero, otherwise it is ignored. When the edge router receives a packet with a non-zero offset, it assumes it is a fragment and the router forwards it as a UDP packet in the domain network by clearing the in-order-bit in the TOS field. Because the 13-bit fragment offset is rarely used, out-of-order delivery of TCP packets should not have serious effect. To prevent fragmentation within domain, the DF field in the IP header is set at the ingress point. Since this solution is applied in a single domain and it is possible to know the MTUs of all the links, the use of the 13-bit offset field can be prevented in the domain. When a packet leaves the domain, the egress router clears the offset field if the in-order-bit is set.

By decoupling the key computation from the source-destination, not only better ran-

domness in the key distribution can be achieved but the per-packet processing at each hop is reduced. The above technique does not in any way prevent other per-hop-behaviors (PHBs) of Diffserv from being implemented in conjunction with minimum-delay routing. If desired, the connection table can be eliminated and CRC16 hash be used on each entering packet instead; even then, the hash computation is done only once at the ingress node. Note that the Diffserv architecture does not prevent using per-connection information at the edge routers various purposes.

In OSPF-OMP, boundary values are associated with each next-hop. When a packet arrives, its hash is compared with the boundary values to determine the next hop. This has $O(N^i)$ complexity. We propose a different method, which uses more memory, but is faster. At node i, a hash table HT_j^i with M_j^i entries is associated with each destination j. Each entry of the table points to a next-hop $k \in S_j^i$. The next-hops are distributed randomly over the range, and the fraction of entries that point to a particular next-hop will be proportional to the routing parameters. That is, for each $k \in S_j^i$, $m_{j,k}^i$ entries of HT_j^i, chosen randomly, are filled with the value k, where $m_{j,k}^i = \phi_{j,k}^i M_j^i$ and $M_j^i = \sum_{k \in S_j^i} m_{j,k}^i$. When a TCP packet for j arrives at i, the mod function on the key is used to index into the table HT_j^i to obtain the next-hop. If M_j^i is chosen such that it is a power of two, the lower $log_2(M_j^i)$ bits of the key can be used to index into the hash table. The complexity of this procedure is constant and is independent of number of neighbors.

3.6.3 Hybrid packet forwarding

Because TCP traffic is allocated on flow basis, there is lack of complete control on the packet forwarding and, the actual traffic forwarded can deviate significantly from the amounts specified by the routing parameters. Fortunately, the skew in the distribution introduced by

```
00. procedure hybrid-forwarding(P)
01. {Executed at i on arrival of packet P for j.}
02. begin
03.       if (P is a UDP packet) then
04.             Let W^i_jk ≤ φ^i_jk W^i_j, for some k ∈ S^i_j;
05.       else if (P is a TCP packet) then
06.             Let P's header map to next-hop k;
07.       endif
08.       Forward P to neighbor k;
09.       W^i_jk ← W^i_jk + sizeof(P);
10.       W^i_j ← W^i_j + sizeof(P);
11. end
```

Figure 3.13: Forwarding TCP packets according to routing parameters.

the hashing mechanism can be ironed out to some extent if there is UDP traffic present. The packets in UDP traffic can be forwarded to neighbors that received too little traffic compared to what the routing parameters allow. The modified packet forwarding procedure is as shown in the Fig.(3.13). Note that OSPF-OMP does not make this distinction between TCP and UDP traffic.

The hybrid procedure is similar to the UDP-only forwarding procedure described in Fig.3.12, except that the skew in distribution created by TCP traffic is mitigated by UDP traffic; the greater the share of UDP traffic in the total traffic, the finer is the distribution of the traffic according to the routing parameters. In the Diffserv model, out-of-order profile packets can be treated as datagram traffic if desired. In section 3.6.4, we will give some performance figures regarding this.

In summary, we believe that the proposed packet forwarding scheme for the architecture can be easily deployed in existing networks, and can be used to implement multipath and routing parameter obtained through other schemes. Though hashing is not new, we combined it with the Diffserv framework and a hybrid packet forwarding scheme to achieve significant performance benefits. Hash computation and comparisons with boundary values at each hop are

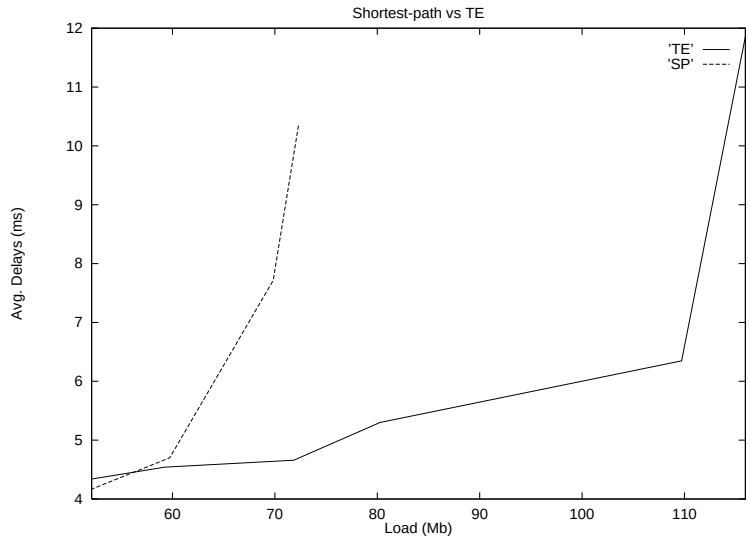


Figure 3.14: Comparison of single path and TE scheme

eliminated, and the proposed architecture can be implemented within the current IP forwarding technologies without needing support from such other forwarding technologies as MPLS. It can also be used to implement other traffic engineering approaches with other optimization criteria [72] as long as the solution can be represented using routing parameters. In the rest of the section, we will refer to this packet forwarding scheme as 'TE'.

3.6.4 Performance of TE Scheme

We tested the effectiveness of the TE scheme through a series of simulation experiments. In each of the experiments, for the same given input traffic matrix and network configuration, the routing parameters are first computed using an off-line minimum-delay routing algorithm and downloaded into the routing tables. Traffic entering the network is then forwarded according to the routing parameters. The end-to-end average delays are measured for each flow and compared under different scenarios: single-path routing, varying volumes of TCP

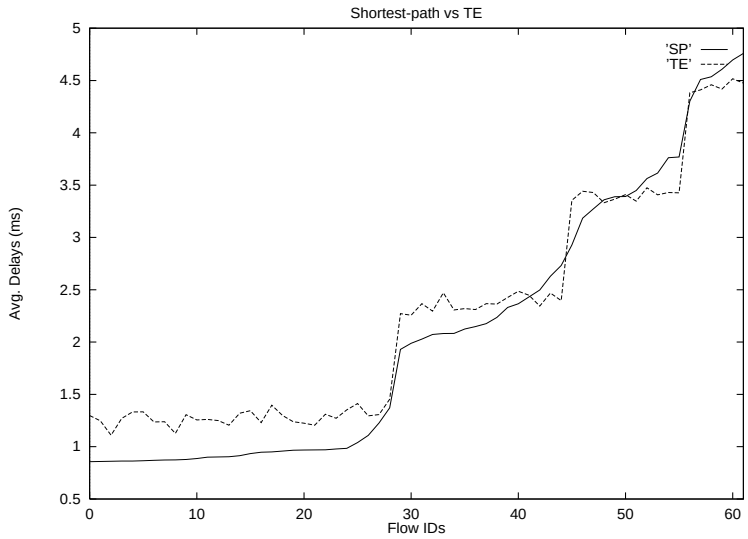


Figure 3.15: Comparison of single path and TE scheme

flows, varying proportions of TCP and UDP traffic.

The network used in the simulation is shown in Fig.3.3. The links have bandwidth 10MB, propagation delay of 100 microseconds and packets are of size 1000 bytes.

Experiment 1: The delays of single-path routing (SP) are compared with the delays obtained in our traffic engineering (TE) scheme. The input traffic consists of 500 identical TCP flows; 50 of them originating from each node. Fig. 3.14 shows the delay versus load comparison. The average delays of single-path routing are denoted by SP and the delays of our scheme are denoted by TE. Observe that the load that the network can carry is much greater in the TE scheme for a given average delay. At very low loads SP performed better because of the tendency of TE to route along longer than shortest paths under low loads. Fig. 3.15-3.17 show the comparison of delays of individual flows for both schemes under three different load conditions. The x-axis in this case denotes flow IDs and the y-axis the average packet delays for the flows. To remove clutter and make the plots clearer, the flows are first sorted in ascending

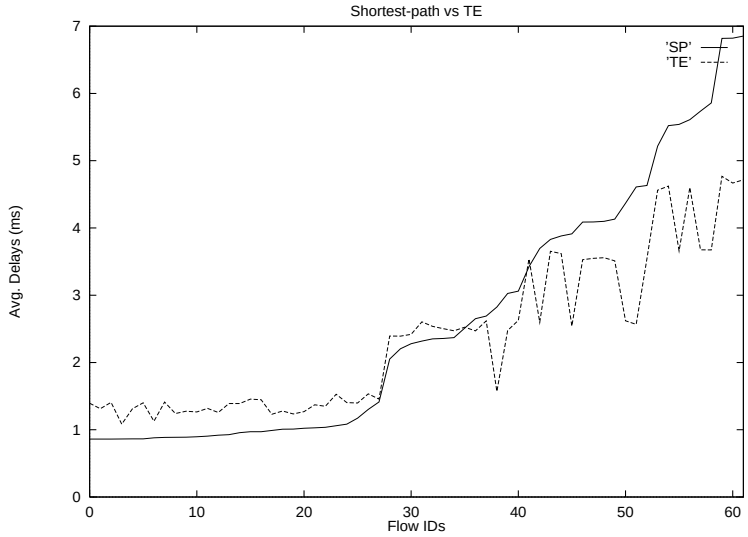


Figure 3.16: Comparison of single path and TE scheme

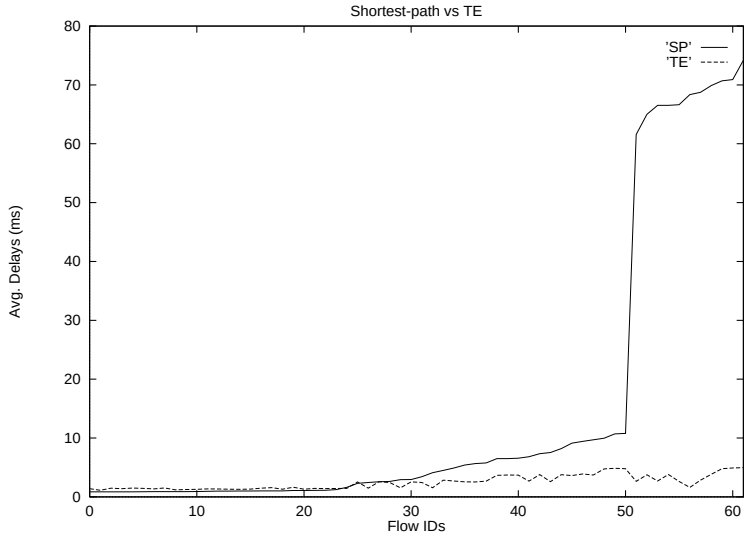


Figure 3.17: Comparison of single path and TE scheme

order of delays of single-path routing and then plotted. As can be seen, the proposed TE scheme significantly outperforms the single-path routing as the load in the network increases.

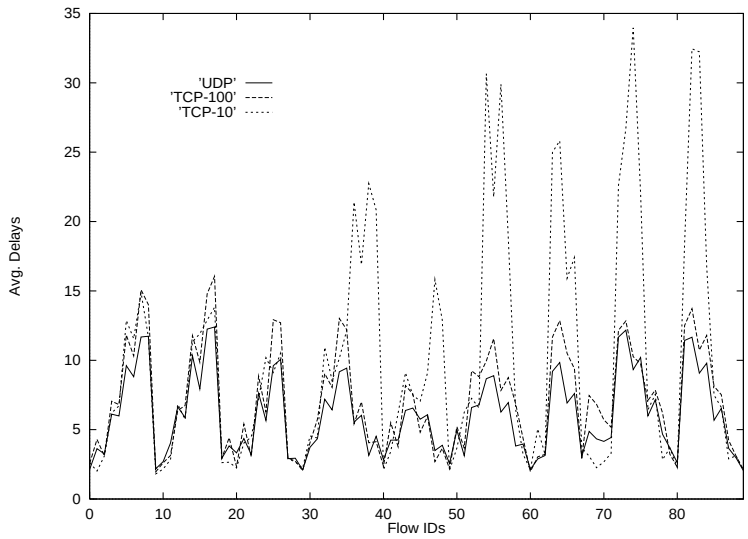


Figure 3.18: (a) Delays as a function of volume of flows

In Fig. 3.15, for low loads, they are very close and often SP is better. However, under high loads, there can be severe congestion in single-path routing which can be avoided in multipath routing as seen in Fig. 3.17. Because of the use of multipaths in the TE scheme congestion and, therefore, the delays are reduced.

Experiment 2: In this experiment, we tested our assumption that delays close to minimum-delays can be obtained when large number of TCP flows are present. TCP-10 represents delays when there are 10 flows between each source-destination pair. UDP represents the optimal routing. Observe that in Fig.3.18 the delays decrease to levels comparable to the optimal as the number of flows between source-destination pair increases from 10 to 100. As explained earlier, this is due to finer granularity in distribution.

Experiment 3: Traffic can be forwarded more accurately according to routing parameters if the presence of UDP traffic and packet sizes are considered. We run the experiment for traffic that consists of 60% TCP traffic and 40% UDP traffic. As expected UDP-40 delays

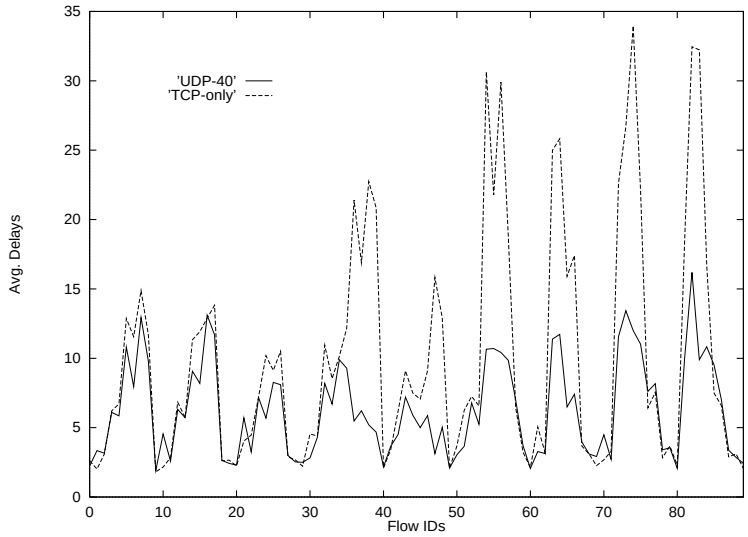


Figure 3.19: Delays in presence of hybrid traffic

are better than TCP-only delays (Fig.3.19).

3.7 Summary

In this chapter, we addressed the traffic engineering problem from a theoretical perspective. Our approach differs from previous approaches in two important ways. First, the framework attempt to achieve near-optimal delay and has strong basis on theoretical results, and second the approach does not use any connection-oriented mechanisms such as MPLS, virtual circuits or any other form of route-pinning to establish paths for packet forwarding. The proposed traffic engineering scheme follows a connection-less paradigm in which multipaths are constructed by distributed routing protocols.

Chapter 4

Multipaths in Guaranteed Services Architecture

Several different architectures have been proposed that support applications requiring strict service guarantees. In the IETF's Integrated Services architecture (Intserv) [10, 76] and other architectures [28, 41, 61], the required bandwidth for each flow is reserved in the network on a *per-flow* basis, and routers ensures that the flows receive their allotted bandwidth by using fair schedulers, such as Weighted Fair Queuing (WFQ) or equivalent [14, 48, 60], at each link. However, these per-flow approaches have many scaling problems as described below.

First, as the links in the backbone networks reach or exceed gigabit capacities, routers are expected to carry large numbers of flows, which requires large amounts of memory to store the routing and reservation state in the routers. Secondly, as the reservation state increases, maintaining the consistency of reservations in the presence of network failures and control message loss becomes complex in per-flow architectures. To maintain the consistency of reservations state, the soft-state reservation protocol RSVP [76] was proposed, which though robust

can be prohibitively expensive due to its use of per-flow refresh messages. Thirdly, as the number of flows on a link grows, the time complexity of link-schedulers (e.g., [48, 60]) grows, thus making it more and more difficult to schedule packets in a timely manner. All these scalability problems arise mainly because the per-flow mechanisms in all these architectures are functions of the number of flows. Another reason, which is fundamental yet rarely emphasized, is that many of the above mentioned architectures are primarily connection-oriented architectures. Because the IP architecture is connection-less, introducing connections in the IP cloud seems to violate the very principle that contributed to the success of IP networks.

The scaling problems of the Intserv architecture are well-known, and are currently being addressed in several fora. There have been many proposals to make RSVP scalable by reducing its refresh message overhead [3, 71]. Unfortunately, these proposals are only partial solutions that mitigate the problem, but do not solve it; the amount of reservation state that needs to be maintained in the routers is still the same. In the aggregated-RSVP [20], the state size is quadratic as all flows between source-destination are aggregated; however, the performance, measured in terms of call-blocking rates, is poor as they are set up along a single-path.

Another approach to providing a deterministic QoS in IP networks consists of eliminating the per-flow reservation state in the core routers and follow a *stateless* approach similar to Differentiated Services (Diffserv) architecture [15]. Some recently proposed architectures [61, 77] follow this approach. They introduce connections through MPLS and dynamic packet state techniques and use a signaling protocol to set up connections. For example, in the SCORE [61] architecture the reservation state is stored in the packets of the flows instead of storing in the routers. The reservation state in the packets is then used by the core routers to estimate the aggregate reservation on the links. There are no explicit refresh messages and thus the

problems associated with lost or delayed refresh messages are greatly diminished. However, bandwidth under-utilization can result from inaccurate estimation of aggregate reservation on the links. Also, the overhead related to state carried in the packets and the processing power required for maintaining consistency of state in the presence of failures, can limit the scalability and efficiency of this architecture. Moreover, the architecture uses a fixed number of shortest-paths between source-destination pairs and does not exploit full connectivity of the network. In the Bandwidth Broker [77] architecture, a central broker is used to store the reservations of all the flows. The scalability problem here arises because the central broker not only has to perform path-selection and termination on per-flow basis but also has to process periodic refresh messages from all the flows in the network if a soft-state mechanism is used to maintain the reservation state. All the above mentioned approaches are *connection-oriented* in nature and as such have inherent limitations in scalability and robustness.

In this paper, we describe a new architecture, SMART (Scalable Multipath Aggregated RouTing), which addresses the scalability and performance problems of the above architectures, and yet approaches the problem from a connection-less perspective. The key idea in the SMART architecture is that flows are aggregated along *multipaths* [67, 69]. Multipaths, as described earlier, are acyclic directed graphs rooted at the destination and are a generalization of single shortest-path routing trees. Flows are aggregated at the ingress router based on class and destination, and in the core routers flows are processed only on aggregate basis. The aggregation is such that, by providing guarantees to the aggregated flow, the guarantees of the individual flows within the aggregate are also guaranteed. The flow classes differ from previous approaches (e.g., [8, 22, 23]) in that they satisfy a *closure property*; the delay bounds obtained for a class are independent of the number of flows that form the aggregate. As a result of the use of multipaths and flow classes, the size of the routing and reservation state in the SMART

architecture is proportional to the number of destinations rather than the number of individual flows; consequently, the soft-state reservation protocol, AGREE [69], used to maintain the reservation state is also scalable. The AGREE protocol is tightly integrated with loop-free multipath routing. Finally, the link schedulers in the SMART architecture process only aggregated flows, which is proportional to the number of active destinations and not the number of individual flows. In summary, the complexity of all the mechanisms in the SMART architecture are a function of the known network parameters, such as number of destinations and classes, which is similar to the complexity in current IP and the emerging Diffserv architecture. Lastly and most importantly, the SMART architecture follows a connection-less approach – a model that contributed to the success of IP. The next section describes the key ideas in the SMART architecture using a fluid flow model, and in the following sections, we extend it to a non-fluid model and give a detailed description of each component of the architecture.

4.1 SMART Overview

We describe the SMART architecture in the context of intra-domain QoS, and accordingly assume a network infrastructure consisting of a single autonomous network running an interior-gateway protocol (e.g., OSPF, IS-IS or the protocls proposed in Chapter 2) that computes distances to destinations from which multipaths to destinations are constructed. For simplicity, we assume a non-hierarchical network. Extending SMART architecture to inter-domain context requires service-level agreements (SLAS) between ISPs that map flow classes in one ISP to flow classes in another ISP. The key idea is to use flow classes that satisfy a closure property and aggregate flows along multipaths. Within the core and gateways only aggregated state is maintained, and only end-hosts maintain per-flow information. The reservations are maintained using a soft-state protocol, AGREE. The flow setup is established in-band.

4.1.1 Flow Aggregation

Let a real-time flow be specified by the tuple $(\sigma,\ \rho)$, where σ is the maximum burst size and ρ is the average bandwidth required by the flow [13, 48]. We assume that each real-time flow admitted into the network is monitored at the entry point to enforce the flow specification, by a token bucket with a bucket size σ and token generation rate ρ. Define the *burst-ratio* of a flow as the ratio of its burst size to the bandwidth rate, that is, $r = \frac{\sigma}{\rho}$. An equivalent way to specify a flow is $(r,\ \rho)$, where r is the burst-ratio and ρ is the average bandwidth as before, because the burst size can always be obtained by $\sigma = r\rho$. A nice property of flows specified using the burst-ratio is that flows with the same burst-ratio can be merged and divided without changing the burst-ratio of the resulting flows! Let a flow be specified by (σ, ρ), then the amount of traffic A that arrives in an interval $[\tau, t]$ for this flow is given by $A(\tau, t) \leq (\sigma + \rho(t - \tau))$ which is, using burst-ratio $r = \sigma/\rho$, equivalent to ([48])

$$A(\tau, t) \quad \leq \quad \rho(r + (t - \tau)) \tag{4.1}$$

A fluid model is used to simplify our analysis and focus on the new concepts presented in the architecture. If this flow is split into two streams, A_1 and A_2, such that stream A_1 gets a fraction ϕ_1 of the flow and A_2 gets a fraction ϕ_2 of the flow, then the arrival pattern for the two output streams is as follows:

$$A_1(\tau, t) \quad \leq \quad \phi_1\rho(r + (t - \tau)) \tag{4.2}$$

$$A_2(\tau, t) \quad \leq \quad \phi_2\rho(r + (t - \tau)) \tag{4.3}$$

Therefore, the two resulting streams A_1 and A_2 can be characterized by the parameters $(r, \phi_1\rho)$ and $(r, \phi_2\rho)$, which implies the two sub-flows have the same burst-ratio. Similarly,

let two flows with traffic profiles (r_1, ρ_1) and (r_2, ρ_2), merge into a single flow. The amount of traffic that arrives in an interval $[\tau, t]$ for the aggregated flow is given by

$$A(\tau, t) \leq \rho_1(r_1 + (t - \tau)) + \rho_2(r_2 + (t - \tau)) \quad (4.4)$$

$$\leq (\rho_1 + \rho_2)(\frac{\rho_1 r_1 + \rho_2 r_2}{\rho_1 + \rho_2} + (t - \tau)) \quad (4.5)$$

Accordingly, the aggregate flow can be characterized using burst-ratio by $(\frac{\rho_1 r_1 + \rho_2 r_2}{\rho_1 + \rho_2}, \rho_1 + \rho_2)$. Note that if $r_1 \leq r_2$, then $r_1 \leq (\frac{\rho_1 r_1 + \rho_2 r_2}{\rho_1 + \rho_2}) \leq r_2$, which implies that the burstiness of merged flow cannot be greater than the burstiness of the more bursty of the two input flows. Therefore, the resulting merged flow can be characterized by $(r_2, \rho_1 + \rho_2)$. The strength of the burst-ratio concept is that characterizing multiplexed and demultiplexed flows becomes tractable; consequently, the delay bounds offered to them by the link schedulers is also simplified.

We now use the burst-ratio to define Q *real-time flow classes.* A burst-ratio R^g is associated with each real-time flow class g, such that $R^{g-1} < R^g$ and R^1 is zero. A flow with specification $(r,\ \rho)$ is classified at the source as belonging to class $g > 1$ if its burst-ratio r is such that $R^{g-1} < r \leq R^g$. Note that there are classes for other non-real-time traffic, such as best-effort; however, we are only concerned with real-time flows in this paper. From Eqs. (4.1)-(4.5) it follows that, if two flows belonging to the same class g are merged, then the resulting flow also belongs to the same class g. Similarly, if a flow belonging to a class g is *split* into two or more flows in fixed proportions, then each flow also belongs to the same class g.

A class identifier g is included in every packet that belongs to a flow of class g. The class identifier is used by the schedulers at the links to perform class-oriented fair-scheduling where all packets belonging to the same class are treated as same flow.

Each link in the network is serviced by a fair scheduler like WFQ, which provides bandwidth guarantees and flow isolation. Instead of servicing each individual flow, the sched-

ulers service a fixed number of queues (Q) that correspond to the flow classes. For simplicity, we assume that all bandwidth on the link is available for real-time reservation. A variable rb_k^i is associated with each link (i,k), which specifies the unreserved or residual bandwidth. For each real-time class g, let B_{ik}^g be the total bandwidth reserved for real-time class g. Then $rb_k^i = C_{ik} - \sum_{p=1}^{M} B_{ik}^p$ is the residual bandwidth, where C_{ik} is the capacity of the link (i,k). When there is no flow on the link $rb_k^i = C_{ik}$.

A consequence of using a fixed number of flow classes is that the complexity of the schedulers at the links is reduced to a constant order. The schedulers now only have to keep the reservation state on an aggregated basis. Only per-flow-class state needs to be maintained, such as the cumulative rate reserved for the class and the time stamps of the last packet belonging to that class. The link schedulers keep no state on a per-flow basis. Packets arriving at the links are aggregated into queues based on the class of the packets, irrespective of their origin and destination. The routing table of a router determines the particular link scheduler the packet enters, and the class label determines the specific queue in the link scheduler that the packet enters. This is far more scalable than architectures in which there is per-flow state. The time complexity of the scheduling decision, as well as the space complexity of the state required to maintain, are now both constant. The admission test for incorporating the resource requirements for a new flow is straightforward — only the availability of the bandwidth on the link and memory in the router need to be checked. The amount of memory for a class g in the scheduler at link (i,k), for lossless delivery, is atmost $R^g B_{ik}^g$ where B_{ik}^g is the bandwidth reserved for class g on the link [48, 13]. In section 4.5 we describe how the bandwidth and memory reservations are made during flow setup.

Aggregating flows removes isolation between flows which, in general, results in increased burstiness in traffic. In section 4.2, we present a simple technique to aggregate flows

that minimizes traffic burstiness and define a small number of aggregated flow classes based on it. Once flows with a particular destination and class are aggregated, they collectively share the bandwidth allocated to that class along the multipath of that destination and are serviced collectively by the routers. Note that the packets of a flow can follow any path in the successor graph in a connection-less fashion; there is no explicit connection maintained on a per-flow basis.

4.1.2 Multipaths

For each destination, a multipath is constructed using the distances computed by the routing protocol. A multipath is an acyclic directed graph with the destination as the sink node and flows of a particular destination are always established along the multipath of that destination. Let D_j^i be the distance from router i to router j measured in number of hops. Define the successor set at a router with respect to a destination as consisting of *all neighbors of a router that are closer to the destination, or are at equal distance but with a numerically lower address.* The successor set of node i for j is denoted by $S_j^i = \{k | k \in N^i \wedge (D_j^k < D_j^i \vee (D_j^k = D_j^i \wedge k < i))\}$, where N^i is the number of neighbors of router i. Now, with respect to a router j, the successor sets S_j^i define a successor graph $SG_j = \{(m,n) | (m,n) \in E, n \in S_j^m(t),\ m \in N\}$, where N is the set of nodes in the network and E is the set of links. A shortest multipath from router i to j is a generalization of shortest path and is defined as the subgraph of SG_j consisting of all nodes that are reachable from the source i. Fig. 4.1(b) shows the shortest multipath with destination 0 for the network in Fig. 4.1(a).

4.1.3 Packet Forwarding

Figure 4.2 shows the data path schematic for a router in the SMART architecture. Router i has N^i links and each outgoing link is serviced by a WFQ scheduler or equivalent.

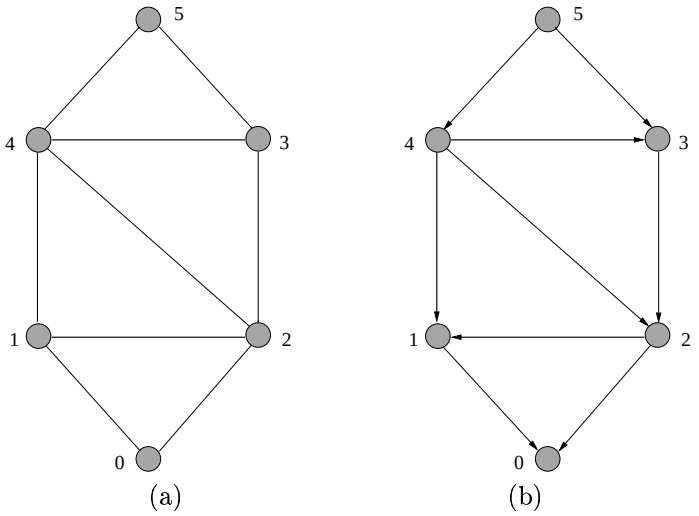


Figure 4.1: (a) A sample network (b) Multipath successor graph

Router i uses token buckets to enforce the rates of the Z flows generated locally by the host attached to the router. It then encodes the destination and class identifier in each packet before it forwards it the next router. At router i, let S_j^i be the set of next-hop routers for destination j. Packets received by router i destined for router j are only forwarded to neighbors in the set S_j^i. Because there can be more than one next-hop in S_j^i, bandwidth for each of the next-hops must be specified. Let $B_{j,g,k}^i$ specify the aggregated bandwidth of class g and destination j that is forwarded to neighbor $k \in S_j^i$, and let $B_{j,g}^i = \{B_{j,g,k}^i | k \in S_j^i\}$. Therefore, a routing table entry is of the form $\langle j, g, B_{j,g}^i, S_j^i \rangle$, where the class g and the destination j uniquely identify a table entry.

When the router receives a packet with destination j and class g, it accesses the corresponding routing table entry and determines a successor k for this packet from S_j^i according to the set $B_{j,g}^i$. This task is performed by the *distributor* (Fig.4.3(b)), which uses a scheduling discipline to allocate packets to successors according to routing parameter set $B_{j,g}^i$. The algorithm for this has been provided in section 3.6.1. The router then puts the packet in the queue

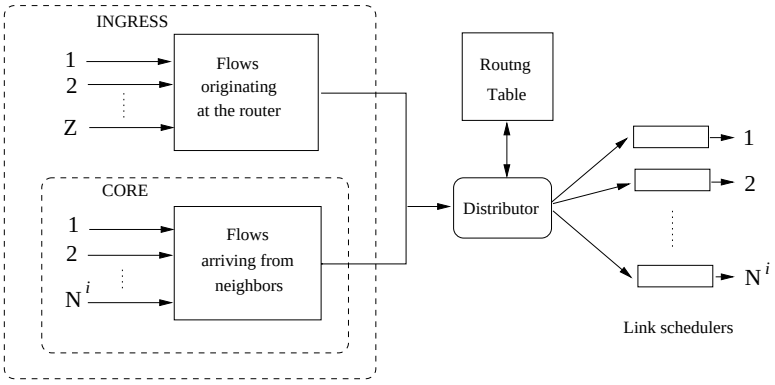


Figure 4.2: Block diagram for the SMART router

of j and class g at the link scheduler of link to (i,k). The time complexity of determining the next hop by the distributor is constant as there are fixed number of neighbors.

4.1.4 Signaling

When flows are added and deleted, the routing tables are modified to reflect the bandwidth requirements as follows. Assume a flow request with class g and bandwidth ρ is required to be established between a particular source and destination j. A path P is selected such that, for each link $(i,k) \in P$, it is true that $k \in S_j^i$ and $B_{j,g,k}^i + \rho \leq C_k^i$, where C_k^i is the link capacity. The routing table entry is then modified such that $B_{j,g,k}^i \leftarrow B_{j,g,k}^i + \rho$. Similarly, the allocated bandwidth is decremented when flows are terminated. The per-flow reservations are only kept at the end hosts; in the core routers and the edge routers connecting autonomous systems aggregated state is kept. Note that the link admission test takes $O(1)$ time which is much simpler, for instance, than the admission tests in [22, 41, 75] which depend on the individual reservations already made to other flows and are generally complex. Also, the link schedulers have to service flows on per-destination per-class basis, irrespective of number of flows through the link. The number of refresh messages on a link used by the soft-state refresh

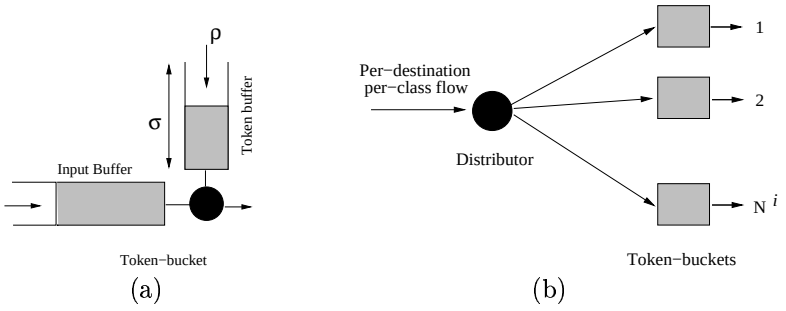


Figure 4.3: (a) Dynamic Token Bucket (b) Distributor

mechanism in AGREE [69] is bounded by $O(NQ)$, where Q is the number of flow classes and N the number of destinations.

4.2 Flow Aggregation: Non-fluid Model

We characterize a flow by the parameters (L, ρ), where L is the maximum size of any packet of the flow and ρ is the average rate of the flow. This is slightly different from the flow parameters in the fluid model of the previous section. Flows are then grouped into classes based on their maximum packet sizes and their rate. Assume that there are Q real-time classes, and with each class g associate a maximum packet size L_g and rate of ρ_g. A flow belongs to class g if the maximum size of its packets is less than L_g and its rate is at least ρ_g. The flow's class is determined at flow setup and when the application sends its data packets. The ingress router inserts the flow's class identifier in the TOS field of the IP packet header before forwarding it to the network. At the ingress routers and in the core routers all flows of the same destination and class are merged and handled as a single flow. We now show how, by providing guarantees to the aggregate flow, the guarantees of individual flows in the class aggregate can be ensured.

Flows are shaped at the ingress node to have at most a single packet burst before

merging with other flows. That is, a flow f with parameters (L, ρ) is shaped with token bucket with bucket size L and token rate ρ (Fig.4.3(a)). If the flow f is serviced by a WFQ scheduler at a link (i, k), the delay bound for a packet of the flow d_f is given by

$$d_f = \frac{L}{\rho} + \frac{L_{max}}{C_{ik}} + \tau_{ik}. \tag{4.6}$$

where τ_{ik} and C_{ik} are the propagation delay and capacity of the link respectively and L_{max} is the maximum packet size allowed for any packet in the network [48]. Now, if the flow belongs to the class g, then $L \leq L_g$ and $\rho \geq \rho_g$ and thus we have $\frac{L}{\rho} \leq \frac{L_g}{\rho_g}$. Therefore,

$$d_f \leq \theta^g_{i,k} = \frac{L_g}{\rho_g} + \frac{L_{max}}{C_{ik}} + \tau_{ik}. \tag{4.7}$$

That is, $\theta^g_{i,k}$ is the delay bound for the class g at link (i, k). We can easily show that the delay bound $\theta^g_{i,k}$ holds for the flow even after it is aggregated with other flows belonging to the same class. Assume that flow f is merged with $n-1$ other flows of the same class g and shaped to a single packet burst. The maximum burst the aggregate flow can have is nL_g. Thus, the resulting aggregated flow can be characterized by the token-bucket parameters (nL_g, ρ^a), where the aggregate bandwidth ρ^a is the sum of rates of the participating flows. The delay bound offered by WFQ to the aggregate is then $\frac{nL_g}{\rho^a} + \frac{L_{max}}{C} + \tau_{ik}$. However, this delay bound cannot be used in the end-to-end delay bound for the flow f, because ρ^a varies in a dynamic environment where flow f may merge with different flows at different times. Given that $\rho^a \geq n\rho_g$ always holds true, a delay bound of $\frac{nL_g}{n\rho_g} + \frac{L_{max}}{C} + \tau_{ik}$ will always hold for the aggregate flow and the constituent flows. This is of course equal $\theta^g_{i,k}$; therefore, for flow f we can use the delay bound of $\theta^g_{i,k}$ on the link in its computation of end-to-end delay bound, irrespective of what flows are aggregated with it and at any time as long as they are of the

same class. Depending on what flows are part of the aggregation tighter delay bounds can be obtained; however, our objective is to obtain delay bound that is independent of constituent flows.

The link scheduler introduces some delay-jitter in the traffic passing through the link. This is removed at the receiving router by reshaping the traffic using token buckets before it is merged with other flows. There is one token-bucket for each class-destination pair at the receiving end of the link. The token buckets are dynamic in the sense that each time a flow is added on the link during flow setup, the parameters of the corresponding token bucket are adjusted. That is, for link (i,k), class g and destination j at node k, the rate of the corresponding token-bucket is set at rate $B^i_{j,g,k}$ and the bucket-size at $\frac{L_g B^i_{j,g,k}}{\rho_g}$. The maximum delay experienced by a packet of class g in the link scheduler and in the shaper at the receiving end of the link is $\theta^g_{i,k}$. The traffic emerging from the shaper belongs to class g and can be readily merged with the shaped traffic of the same class received on the other links.

Because the distributors cannot split packets, an extra burst is introduced by the distributors, which has to be incorporated in the end-to-end delay bounds. This is removed using a token-bucket shaper as shown in Fig 4.3(b). There is one token-bucket shaper for each class-destination pair at the output queue of the distributor. The rate of the token-bucket shaper at link (i,k), for j and class g is set at $B^i_{j,g,k}$ and the bucket-size to the $\frac{L_g B^i_{j,g,k}}{\rho_g}$, and is adjusted as flows are setup and terminated. The extra burst can be at most L_g and the proof for this is given in [70]. The delay introduced by the shaper k of the distributor output is $\frac{L_g}{B^i_{j,g,k}}$ which is upper bounded by $\frac{L_g}{\rho_g}$ as $B^i_{j,g,k} \geq \rho_g$.

The number of queues at a link scheduler is $O(|N|Q)$. This can be reduced to $O(Q)$ by merging all flows of a particular class, irrespective of the destination, into a single queue. Because flows have different destinations the per-destination aggregated flows may have to

be extracted from the per-class aggregate at the receiving end of the link. Merely reshaping the class-aggregated flow to fit the class envelope is not sufficient in this case, and the class-aggregated flow must be restored completely to the traffic pattern it had before entering the scheduler. To achieve this, instead of using a token-bucket for each class-destination pair at the receiving end of the link, a device similar to the *regulator* proposed by Zhang et al [75] is used for each class-aggregated flow. The regulator works as follows. If the delay in link scheduler is θ for a packet, the regulator holds the packet for time $(\theta^g_{i,k} - \theta)$ before forwarding to the distributor. By delaying each packet passing through the link to experience the worst-case delay of $\theta^i_{i,k}$, the regulator restores the class-aggregated flow to the form it had before it entered the link scheduler. Now the packets of a particular destination and class can be extracted from the class-aggregated flow and freely merged with traffic of the same destination and class received on the other links. The disadvantage of this scheme is that a delay field must be used in the packet which makes the scheme difficult to implement in current IPv4 architecture.

Aggregation in the context of guaranteed flows has been discussed before [8, 23], but they primarily deal with static aggregation of flows between a source-destination pair, and do not address delay guarantees when individual flows enter and leave the aggregation dynamically. In the aggregation scheme proposed by Guerin [22], the delay bounds are dependent on the constituent flows of the aggregated flow. The closure property differentiates our aggregation method from the other aggregation methods. The aggregation scheme proposed for RSVP [20] is meant for aggregating reservation state of flows within a single multicast group. Our aggregation scheme is orthogonal to the aggregation achieved in RSVP. In our architecture, aggregated flows are shaped and not individual flows as in [34, 41], in which the benefits of per-hop shaping can be realized (e.g., reduction in buffer sizes), but are largely undone by the

per-flow traffic management that must simultaneously be employed!

The main drawback of using multipaths, however, is that packets may arrive out of order at the destination, but this should not be a concern, if deterministic end-to-end delay bounds are also provided concurrently. Typically, real-time applications use a playback time, which means packets need only arrive within a specific time frame, and arrival order of the packets does not matter.

4.3 Enhanced Multipaths

Using the shortest multipath as defined above poses a problem: when two routers are equidistant from the destination, ties are resolved based on addresses, which may result in an unfair and uneven successor graph. Fig. 4.4(a) shows an example of such an uneven successor graph with 4 as the destination. The longest path from 5 to 4 is four hops and the shortest path is one hop. Because end-to-end delays are determined by the longest path (Eq.(4.8)), this results in longer delay bounds for flows from E to A. Furthermore, the link (1, 4) is a hot-spot link, because all traffic reaching 1 must be transmitted on link (1, 4) as there is no alternative.

This unevenness can be fixed in a couple of different ways. In the first method, the successor set is restricted to consists of *all neighbors that are strictly closer to the destination than the node itself.* That is, $\hat{S}_j^i = \{k \mid k \in N^i \wedge D_j^k < D_j^i\}$, which is in fact the next-hop set computed by OSPF. A shortcoming of this approach is that it does not use the full connectivity of the network. For example, in Fig. 4.4(b), which is a successor graph using $\hat{S}_j^i$ for destination 4 in the network given in Fig. 4.1(a), some links are not used. This results in lower utilization of network bandwidth and higher call-blocking rates.

In the second method, the successor set is defined as $S_j^i = \{k | k \in N^i \wedge D_j^k \leq D_j^i\}$. That is, the successor set with respect to a destination consists of *all neighbors of a router that*

are closer to or at the same distance from the destination. We call this the *enhanced multipath* (EMP). For example, Fig. 4.4(c) shows the EMP for destination 4 in Fig. 4.1(a). The EMPs fix the unevenness problem of the shortest multipath and also improve the bandwidth utilization over the multipaths computed by OSPF. However, there is one important problem with EMPs that needs to be addressed. Note that each router now includes the neighbor that is equidistant from the destination in its successor set, which can cause packets to loop. This can be easily fixed using the following simple technique. Each packet carries a bit-flag called the *e-bit*, which indicates whether the packet was ever forwarded to a *peer* router (neighbor router that is at the same distant from the destination) on its path so far. A router forwards a packet to a peer neighbor only if the e-bit is set; otherwise, the packet is forwarded to one of the *subordinate* neighbor (neighbor whose distance is strictly less than the distance of this router). If a packet is forwarded to a peer neighbor the e-bit is cleared so that all the future routers visited will forward it only to their subordinate routers. It is easy to see that a packet can be forwarded to peer neighbor at most once, thus preventing packet from looping. The packets of a flow can, therefore, follow at most $(D_j^i + 1)$ hops. The ingress router sets the e-bit of the packets only for those flows that were established along the EMP with length $(D_j^i + 1)$. The per-flow e-bit information of the flows is maintained only at the ingress router. The routing table entries are extended to specify the bandwidths for traffic with e-bit set and for traffic without e-bit set. The extended routing table entry is $\langle j, g, S_j^i, B_{j,g}^i, \hat{S}_j^i, \hat{B}_{j,g}^i \rangle$. When a packet is received with the e-bit set the distributor uses the $B_{j,g}^i$ to determine the next hop, otherwise the $\hat{B}_{j,g}^i$ is used. As already mentioned, the e-bit is cleared before the packet is forwarded to a peer.

Our intuition for using EMPs follows from the dynamics of Widest-Shortest path-selection strategy which is often used as a benchmark [21]. The Widest-Shortest path-selection algorithm first selects valid paths that are the shortest, and as bandwidth on the shorter paths is

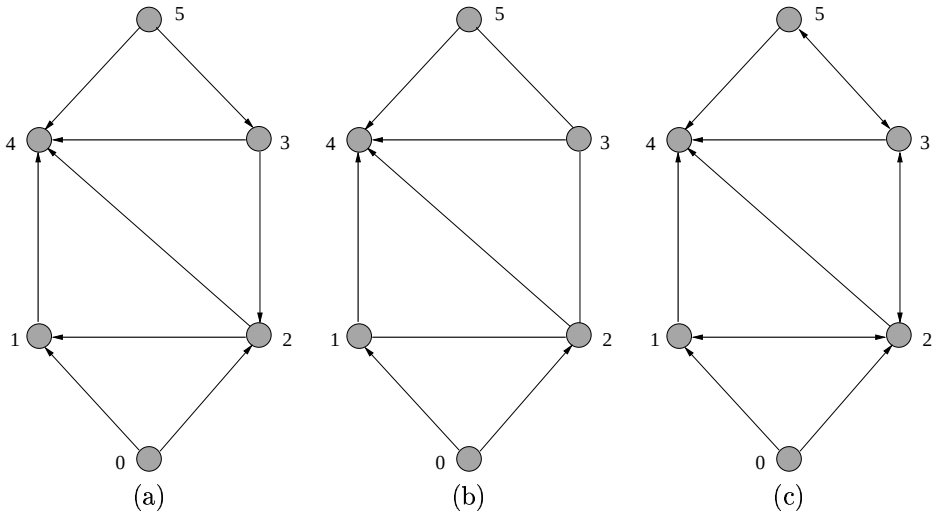


Figure 4.4: (a) An uneven multipath (b) Equal cost multipath (c) enhanced multipath

consumed, longer paths are tried. Effectively, the requests are first attempted along the EMPs. By always selecting paths along EMPs, bandwidth utilization comparable to Widest-Shortest path can be attained [67].

4.4 End-to-End Delay Bounds

4.4.1 Multipath Flows

The end-to-end delay bound for flows established along multipaths must include the delays experienced in the distributor and the delays in the link schedulers. Because the links can have different bandwidths, the delay of the worst path in the multipath should be chosen for computing the end-to-end bound. Thus, for a class g flow from router i to j, the end-to-end delay $\delta^i_{j,g}$ is recursively defined as follows

$$\delta^i_{j,g} = \frac{L_g}{\rho_g} + MAX\{\theta^g_{i,k} + \delta^k_{j,g} \mid k \in S^i_j\}. \quad (4.8)$$

where $\theta^g_{i,k}$ is as in Eq.(4.7) and $\delta^j_{j,g} = 0$ for all j and g. The first term on the right hand side of Eq.(4.8) is due to the delay in the distributor. Observe that the end-to-end delay bounds in this architecture can be determined at the ingress node itself using the class and destination of the flow and the link information propagated by the routing protocol. Note that we assume the end-to-end delay bounds are known a priori and are provided by the network through design; they are not specified by the user. For high-speed backbone networks in which the link capacities tend to be very high compared to individual flow bandwidths, the ratio $\frac{L_g}{\rho_g}$ will be the dominant component of Eq.(4.7). So the $\delta^i_{j,g}$ of Eq.(4.8) reduces to

$$\delta^i_{j,g} = (2M^i_j - 1)\theta^g \quad (4.9)$$

where $\theta^g = \frac{L_g}{\rho_g}$ and M^i_j is the longest path from i to j in the shortest multipath. For an EMP $M^i_j = D^i_j + 1$, and the end-to-end delay-bound is

$$\delta^i_{j,g} = (2D^i_j + 1)\theta^g \quad (4.10)$$

The error terms ($L_{max}/C_{i,k}$ and $\tau_{i,k}$) due to link capacities and propagation delays can be propagated using the routing protocol. The ingress router simply needs to add this to the end-to-end class delay.

4.4.2 Single-path Flows

The delay-bound $\frac{L_g}{\rho_g}$ introduced by the distributor can be expensive for flows that have small bandwidth. The resulting end-to-end delay-bound can be as much as twice the

delay bounds of a per-flow architecture. This, however, can be overcome by simply bypassing the distributor, and forcing all packets of a flow to follow the same path. The flows are still established along multipaths, but flows of the same destination do not share the bandwidth along the multipaths. We have shown how this is implemented in Section 3.6.2. The packets carry a key which is used to hash into one of the subordinate next-hops. The ingress router maintains the key information in the per-flow table. In the core routers, there is no need for distributing the packets of these classes along the multiple next-hops and thus it eliminates the delay introduced by the distributors, and the delay bound obtained will then be comparable to those provided by per-flow mechanism. If a single-path flow of class g and path P from i to j, then the maximum end-to-end delay-bound $\delta^i_{j,g}$ for this flow is given by

$$\delta^i_{j,g} = \sum_{(m,n)\in P} \theta^g_{m,n} \tag{4.11}$$

Again, assuming that class delays are the dominating components at all links, we have

$$\delta^i_{j,g} = D^i_j \theta^g \tag{4.12}$$

Note that the delay bound obtained through Eq.(4.12) is half that of the delay-bound obtained through Eq.(4.10). The disadvantage of using single-path flows is that bandwidth utilization can be lower compared to the utilization achieved through multipaths.

4.5 Quality of Service Routing

QoS routing refers to the task of determining a path in the network that satisfies the required characteristics of the flow such as delay, bandwidth, delay-jitter and loss probabilities. It is well-known that the QoS routing problem with multiple constraints is NP-complete in

the general case. For example, finding a path that satisfies the delay-bound as well as loss probability is NP-complete [73]. However, in most practical situations bandwidth and delay are the most important constraints and, fortunately, for this case, the QoS routing problem is polynomial. In addition, if delay-jitter is correlated to delay, the QoS routing problem with bandwidth, delay and delay-jitter constraints is polynomial [50]. The widest-shortest path (WSP) algorithm and the shortest-widest path (SWP) are two well-known algorithms for finding a path with bandwidth and delay constraints. The WSP is often used as the path-selection algorithm which is fast as well as achieves some load-balancing of the requests by not consuming too much bandwidth on any one link.

Despite the polynomial complexity of these path-selection algorithms, the complexity of QoS routing task can be prohibitively expensive due to the intricacies of the underlying architecture. In the IETF proposed Intserv architecture, the link characteristics and available bandwidth on the links are advertised periodically to all the routers. Based on this information, each router upon arrival applies the path-selection algorithm upon arrival of a request to determine the desired path. However, there are many scaling problems with this basic approach. Firstly, the link bandwidth advertisements can be a substantial overhead in terms of broadcasting and processing them. Secondly, the invocation of the path-selection algorithm, such as WSP, for each request can be prohibitively expensive. Several optimizations have been proposed to address these limitations [17, 58]. These techniques only alleviate the scaling problems, and moreover, whether they justify the performance gains is still debatable.

A promising strategy is to have the on-demand path selection depend only on the bandwidth constraints. Often, substantial gains can be achieved by avoiding link advertisement and using only the local information. For example, in one approach a set of k paths is be maintained at the source, and a path is selected from the set based on the recent success rate

of the paths [26]. These paths are typically establishing using connection-oriented techniques and as a result the whole path is committed to the flow by the decision at the source. Our approach is similar in spirit as it relies only on local data, but does not maintain the k shortest paths and does not perform any local on-line measurements. In the SMART architecture, at each router the next-hop is chosen based on the available bandwidth of the outgoing links. In SMART, the network specifies the delay bounds available to the applications based on the class and destination; the applications do not specify the delay bound. The different classes that are available at each routers are handled at the design time.

4.5.1 Path Selection Schemes

The ESG scheme described below is the heuristic we propose for the SMART architecture. When an application makes a request of the form $(j,\ g,\ \rho)$ at the source router, where j is the destination and ρ is the bandwidth of the flow and g is the class of the flow, the source initiates the flow setup in which path-selection and bandwidth reservation proceed in tandem using only adjacent link information at each hop. The heuristic for choosing the next hop from the successor set can use any strategy.

- ESG Scheme: The bandwidth requirement of the requested flow is signaled along a single path in the network. The widest outgoing link (i.e., link with largest residual bandwidth) in the successor set $\hat{S}_j^i$ is chosen at each hop during the signaling process.

When a session is terminated, the ingress router initiates an explicit flow tear-down procedure by issuing a message of the form $(TERM, j, g, \rho)$. Releasing reservations for flows abnormally terminated is discussed in Section 4.6. A next hop is selected from the $\hat{S}_j^i$ which has the required bandwidth to release. Because a flow tear-down is performed only after the flow establishment such an outgoing link can be found. For $k \in S_j^i$ so found, a release request

of the form $(TERM, j, g, \rho_k)$ is forwarded to router k. The same steps are repeated at the receiving routers. The process continues until the release message reaches the destination. The destination simply discards a release message. A heuristic can be used to determine a ρ_k, for each $k \in S_j^i$, such that $\sum \rho_k = \rho$ and $B - \rho_k \geq B_{min}$. For each $k \in S_j^i$, the bandwidth ρ_k is released from the link to k and the routing table entries are updated.

When a large number of high-bandwidth requests are issued, they tend to cause bandwidth fragmentation in the network which increases the overall call-blocking rate. Bandwidth in the network is said to be fragmented if a flow is rejected because there is no single path that provides the requested bandwidth, but there are two or more non-identical paths that collectively provide the bandwidth. By dividing a single flow into several small flows and establishing them independently, fragmentation of bandwidth can be reduced in the network.

- ESG-Mn Scheme: The flow request is first divided into n small flows of equal size and each of these n flows is independently setup using the ESG scheme.

We illustrate through simulations how the ESG-Mn significantly improves call-blocking rates.

4.5.2 Call Blocking Rates

Call-blocking rate is the metric that is used to measure the performance of the path-selection algorithms and is defined as the percentage of requests that are rejected by the network [43, 58, 18]. The proposed schemes described in the previous section are compared with the following schemes with respect to call-blocking rates.

- WS Scheme: Among the feasible paths, the path that has the shortest length is chosen and, if more than one such path is available the one that offers the widest bandwidth is chosen [43].

- WS-BB Scheme: We assume that link-state information propagates instantaneously throughout the network, that is, each router has the most current information regarding the links. This scheme constitutes of course only an upper bound on the performance of the WS scheme.

- SSG Scheme: This is the same as ESG, except that the successor set is restricted to S_j^i.

- SP Scheme: The request is always established along a single shortest path between the source and destination.

Flow requests are generated and signaled using the above schemes and the blocking rate is measured. Comparing call-blocking rates of different schemes is difficult, because it depends on many factors such as arrival pattern of requests, the duration of the calls, the bandwidth request sizes. In the case of WSP, the call-blocking rates also depends on link bandwidth advertisements. To normalize these effects we use the following strategy. The call-blocking rates are obtained as a function of $\rho = \frac{\lambda}{\mu}$, where λ is the rate of arrivals and $1/\mu$ is the mean holding time of the flows. Flow requests have a uniform distribution across the network, that is, the source and destination are randomly chosen with uniform distribution and at each source arrive with an exponential distribution. The bandwidth size of the request is C/β, where C is the capacity of a link and β is a number greater than 1. For the WSP scheme, the update period is set at θ times the inter-arrival time. The experiments are performed on the topology shown in Fig. 2.13.

In Fig.4.5, WSP-Tθ represent the call-blocking rates for WSP with θ update rate. Observe that the performance of ESG scheme is comparable to that of WSP schemes. This shows that link bandwidth advertisements and subsequent application of WSP path selection algorithm does not offer any advantage. Simple next-hop selection based on local-metric performs remarkable well. Also observe that the schemes SSG and SPF which have similar

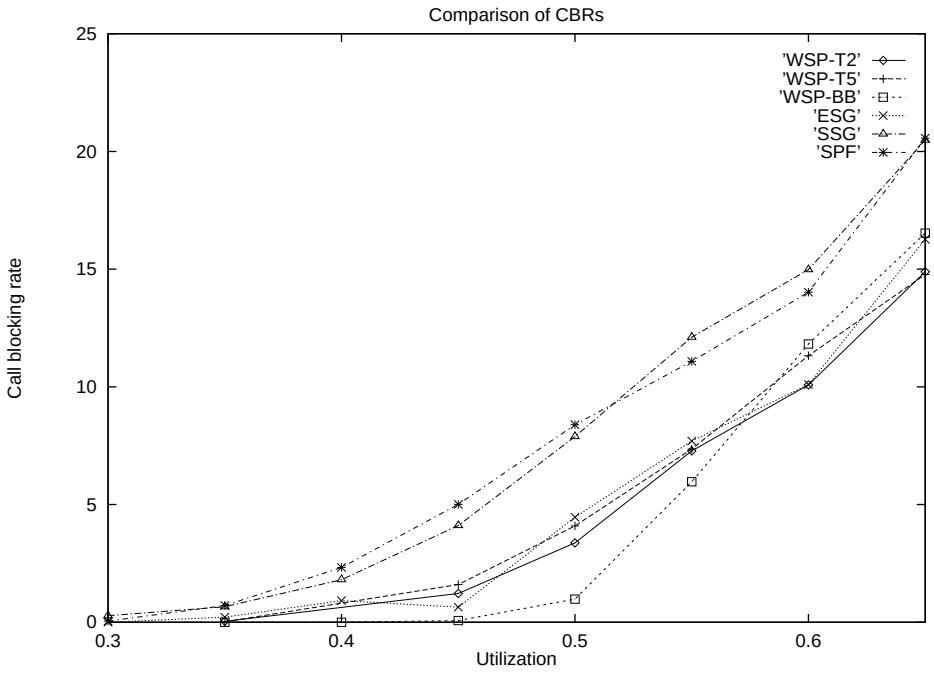


Figure 4.5: Call-blocking rates as a function of ρ. $\beta = 20$

complexity as ESG under-perform substantially compared to ESG.

Fig.(4.6) shows the effect on call-blocking rates when a given flow request is divided into two requests with half the bandwidth each. Due to reduction in fragmentation, the performance significantly increased for both ESG and WSP as indicated by the plots ESG-2 and WSP-2. Also note that ESG-2 has better performance than WSP. While the state size in the routers required for ESG-2 is same as that required for ESG, implementing WSP-2 requires the twice the amount of state in the routers and twice the amount of processing power. However ESG-2 requires twice the number of messages for setting up a flow, but this is also the case with WSP-2. The basic result is that the throughput of the network increases when flows are broken down into small flows due to the better use of network bandwidth, and this benefit can be easily achieved with the ESG scheme when compared to the WSP scheme.

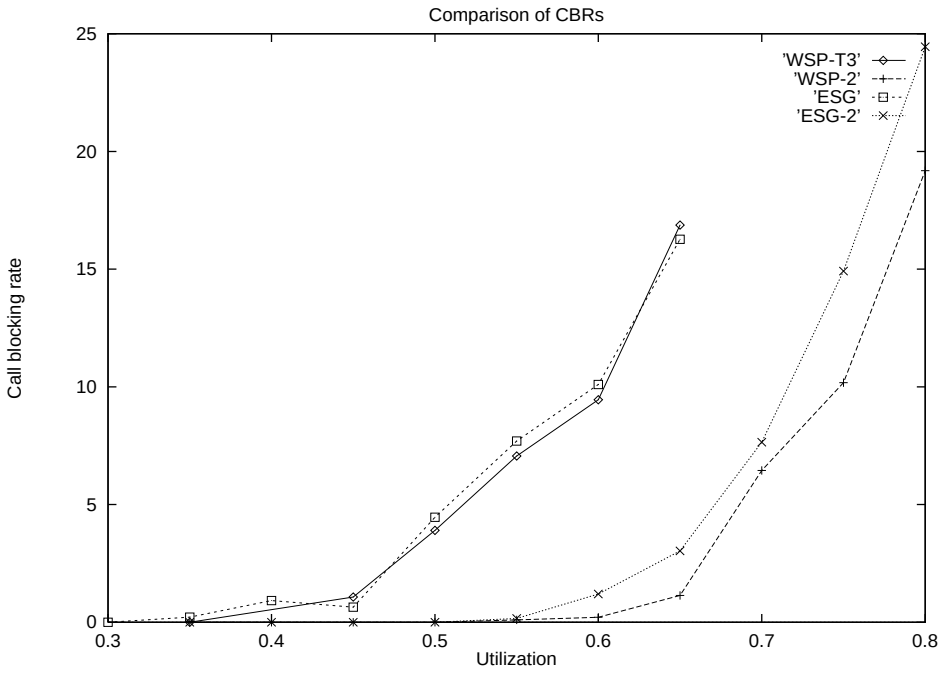


Figure 4.6: Call-blocking rates when flows are divided into several smaller flows

Current architectures such as Intserv already have high state size and establishment of multiple flows can only exacerbate the scaling problem further. So using this approach in traditional architecture is not a viable option. Multipath architectures with routing parameters are better suited to reduce bandwidth fragmentation.

4.6 Reservation Maintenance Protocol

The goal of the AGREE protocol is to maintain the consistency of reservations. If at each router i for all destinations j and classes g, $\sum_{k \in S_j^i} B_{j,g,k}^i = \sum_{k \notin S_j^i} B_{j,g,k}^i + I_{j,g}^i$, then reservations are said to be in consistent state. The pseudo-code of AGREE is shown in Fig. (4.7). The AGREE protocol uses soft-states like RSVP and YESSIR, but because reservation state is on a per-destination per-class basis, *its reservation refresh messages are on a per-destination*

per-class basis. Every T_R seconds (refresh time period), for each destination j and g, the router i invokes $AGREEEVENT(TIMEOUT, j, g, -, -)$ (the parameters b and k are ignored) for comparing the cumulative reservations of the incoming refresh messages with the current reservations and sending its own refresh messages. A refresh message specifies the destination j, class g and the associated bandwidth b. The source node of each flow sends its refresh messages to the ingress node every T_R seconds, stating its destination, class and its rate. At the ingress node all refresh messages of a particular destination and class are aggregated and a single refresh message is sent to the next-hop. When a flow terminates, the source stops sending its refresh messages and the bandwidth reserved for the flow is eventually timed out and released in the network. The core refresh cycle is shown on lines 02-13. Let the reserved bandwidth on the outgoing links in S_j^i for class g add up to bw. Let the refresh messages received by router i for destination j from neighbors not in S_j^i and refresh messages originating at the router, during the previous refresh period add up to a total bandwidth of bt. Note that the refresh messages originating at the router itself add up to $I_{j,g}^i$. First, bt is compared with bw and if $bt = bw$, the reservations are in consistent state and no bandwidth need to be released. The router simply sends a refresh message to each next-hop $k \in S_j^i$ with current allocated bandwidth $B_{j,g,k}^i$.

Reservations can become inconsistent, i.e., $bt \neq bw$, because of flow terminations, link failures and control message losses. To correct the inconsistencies we consider the two separate cases: (1) $bt < bw$, and (2) $bt > bw$. The first case is handled by lines 6-9 and 12 of the pseudo-code. The total incoming bandwidth bt is first divided into b_1,..,$b_{S_j^i}$ such that for each $k \in S_j^i$, $b_k \leq B_{j,g,k}^i$, and then for each $k \in S_j^i$, $B_{j,g,k}^i$ is updated with b_k and a refresh message is sent to k with the new bandwidth b_k. The second case, i.e., $bt > bw$ is generally more difficult, and requires forcing the upstream routers to reduce their outgoing bandwidth.

We describe two techniques to correct this inconsistency. The first method uses the fact that the underlying routing protocol (such as OSPF) informs all routers about link failures. When a router learns about a failed link, it terminate all flows that use that link. The soft-state refresh mechanism will then eventually release the bandwidth reserved for these flows using the same process outlined to handle case (1). A router only needs to remember the path of each flow that originates from it.

The second method uses diffusing computations [27] to correct the inconsistencies. The router i invokes $AGREEEVENT(RELEASE, j, g, k, B^i_{j,g,k})$ for each j and g, when it detects failure of adjacent link (i, k) it invokes

The router updates $B^i_{j,g,k}$ (line 15) and invokes $DIFFCOMP$ if it is in PASSIVE state. The $DIFFCOMP$ procedure first terminates as many flows as possible at the router, and if there is still some bandwidth that must be released to restore consistency, the router distributes the excess bandwidth among upstream neighbors and requests them, using RELEASE messages, to reduce sending required traffic to this router (lines 31-34). The router then enters ACTIVE state indicating that it is waiting for the upstream nodes to reply with ACK messages. When an upstream router receives a RELEASE it repeats the same process. When a router is in ACTIVE state, if it receives RELEASE messages from successor nodes, it immediately sends back an ACK message (line 18). After all ACK messages are received, it transitions to PASSIVE state (line 21) and if the transition to ACTIVE state was triggered by a RELEASE message from the downstream message, it sends the ACK message to the successor node that triggered the transition to ACTIVE state (line 22). When flow-setup and terminate messages are received, they are simply forwarded to the next hop after the reservations are modified.

During routing-table convergence, stray release messages may arrive from current

```
01  procedure AGREEEVENT (type, j, g, k, b)
02  if (type = REFRESH), B^i_{j,g,k} ← B^i_{j,g,k} + b;
03  if (type = TIMEOUT), then {
04      bt ← Σ_{k∉S^i_j} B^i_{j,g,k} + I^i_{j,g};
05      bw ← Σ_{k∈S^i_j} B^i_{j,g,k};
06      if (bt < bw), then {
07          Divide bt into b_k such that Σ b_k = bt and b_k ≤ B^i_{j,g,k};
08          B^i_{j,g,k} ← b_k;
09      }
10      if bt > bw and state^i_{j,g} = PASSIVE, then
11          call DIFFCOMP (j, g, bt − bw);
12      for each k ∈ S^i_j, send [REFRESH, j, g, k, B^i_{j,k}];
13  }
14  if (type = RELEASE), then {
15          B^i_{j,g,k} ← B^i_{j,g,k} − b;
16          if (state^i_{j,g} = PASSIVE), then
17                call DIFFCOMP (j, g, b);
18          otherwise send [ACK, j, g] to k;
19      }
20  if (type = ACK and last ACK message for j and g) {
21          state^i_{j,g} ← PASSIVE;
22          send [ACK, j, g] to s, if s is waiting for ACK;
23      }
24  if (type = SETUP) then // s is the successor on the path
25      B^i_{j,g,k} ← B^i_{j,g,k} + b; B^i_{j,g,s} ← B^i_{j,g,s} + b;
26  if (type = TERMINATE) then
27      B^i_{j,g,k} ← B^i_{j,g,k} − b; B^i_{j,g,s} ← B^i_{j,g,s} − b;
28  procedure DIFFCOMP (j, g, b) at node i
29      if (b ≤ I^i_{j,g}), then terminate flows for j and
30          class g that add up to at least b and return;
31      br ← b − I^i_{j,g};
32      Divide br into b_k and k ∉ S^i_j such that
33          Σ b_k = br and b_k ≤ B^i_{j,g,k};
34      for each k ∉ S^i_j, send [RELEASE, j, g, b_k] to k;
35      state^i_{j,g} ← ACTIVE;
```

Figure 4.7: Event Handling in the AGREE protocol

upstream nodes. These are safely ignored by immediately sending ACK messages even when the router is in PASSIVE state. Similarly, the refresh messages received from downstream nodes and duplicate refresh messages are ignored. When a neighbor k is added or removed

from a successor set, the corresponding $B^i_{j,g,k}$ are reset for each j and g. Note that though we did not explicitly state in the pseudo-code, before initiating the diffusing computation, an attempt can be made to reserve the required bandwidth through a new request and only when the request fails the diffusing computation can be triggered.

The AGREE protocol can be said to work correctly if, after a sequence of link failures and refresh message losses, and if no new flows are setup and terminated, all reservations reflect a consistent state within a finite time. For correct functioning of the protocol, we assume messages on a link are received and processed in order. This prevents race conditions between flow setup, terminate, refresh and release messages. Because the topology stabilizes within a finite time and the routing protocol ensures that loop-free shortest paths are established for each destination, all diffusing computations terminate and all routers return to PASSIVE state for each class-destination pair. In the AGREE protocol, the release messages and the refresh messages only decrease the reserved bandwidths. Because bandwidths cannot decrease forever, no new diffusing computations will be initiated after a finite time. At this time, the bandwidth specified by refresh messages at each node for a particular destination can only be less than or equal to the reserved bandwidth at that node, otherwise this will again trigger another diffusing computation. If on the other hand refresh messages specify lower bandwidth than reserved bandwidth, then that extra bandwidth is eventually released by the usual timeout process of case (1). Therefore, all reservations must eventually converge to a consistent state.

In each refresh period at most $O(QN)$ refresh messages are sent on a link irrespective of number of flows in the network. Because the bandwidth requirements for refresh messages is known a priori, they can be serviced through a separate queue in the link scheduler and guarantee a delay bound. Therefore, refresh messages are never lost due to queuing delays. This is not possible in per-flow architectures as the number of flow on a link cannot be determined

a priori. In AGREE, messages can only be lost due to link failures, which in backbone network is relatively rare. Even then the AGREE protocol is more resilient to refresh message loss compared to a per-flow architecture. In per-flow architectures *a lost refresh message cannot be distinguished from flow termination* and the router interprets a lost refresh message as a flow termination and attempts to release bandwidth from downstream links. In the following cycle, when the refresh message is received correctly, it tries to recover the released bandwidth. In contrast, in AGREE a link can simply *use a null refresh message when it does not carry any traffic for a particular destination and class.* This enables distinguishing flow termination from refresh message loss. When a periodic refresh message is lost, the receiving node recognizes it and continues to use the contents of the refresh message of the previous cycle. In the following cycle, if a refresh message is received correctly, the new refresh message is used. In essence, *refresh messages are sent irrespective of the presence of flows in a synchronous manner* which is only possible because AGREE's reservation state is based on network parameters. This model is scalable, because the worst case bounds on state size depend on the number of active destinations and classes rather than the number of individual flows. For simplicity, we have presented AGREE with respect to a multipath, but it can be easily extended to handle enhanced multipaths.

4.7 Related Work in Resource Management

The scalability problem of the RSVP protocol is well-known and there have been several proposals to reduce its refresh message overhead [3, 46, 47, 71]. The technique in [71] applies a compression algorithm (CRC-32 or MD5) on the reservation state to produce a digest and refresh messages are exchanged for digests rather than for individual sessions. As a result the number of refresh messages is proportional to number of neighbors rather than the number

of sessions. Though the number of messages is significantly reduced, new complexities are introduced in the refresh mechanism. The computation of digests requires grouping of sessions according to next-hops and is expensive when sessions are short lived and individual sessions change paths. Also, neighbors may end up in inconsistent state when the messages do not represent the digests and it is not easy to recover from it. It does not fundamentally reduce the amount of state information that needs to be maintained, and refresh message loss cannot be distinguished from flow termination. Our method differs in that the state is significantly reduced through aggregation of flows, so much so that it depends on network parameters and it is feasible to send per-class per-destination refresh message even when there is no corresponding flow on the link, and help distinguish flow termination from refresh message loss. Moreover, these compression techniques, if desired, can also be incorporated into our approach to further enhance it. In [3], several refresh messages are bundled into a single message. This technique provides scaling benefits, but it compromises on error recovery properties. To recover from corrupted internal state, standard refresh messages must be sent in addition to bundled messages, adding complexity to the protocol and its configuration. However, like the previous technique, this technique does not fundamentally reduce the state size and if needed can be incorporated into our AGREE protocol.

Another approach to reducing refresh message volume is to control the refresh interval [46]. This technique also does not fundamentally decrease the state size, and its focus is on reducing the refresh messages. This technique can be used orthogonal to our technique. The YESSIR protocol [47] uses a sender-initiated approach and avoids separation of reservation and path-finding messages. As a result the processing and protocol complexity is reduced. The AGREE protocol is sender-initiated and has similar benefits, but differs in that it manages reservations that depend on network parameters and is closely tied to the proposed multipath

framework. Lastly, the multipath reservation maintenance feature of the AGREE protocol can be incorporated into YESSIR and RSVP as extensions or as a separate protocol and is a subject of future work.

The Diffserv architecture [15, 25], proposed to address the scalability of the Intserv architecture, uses no per-flow state in the core routers, but the approach is mainly targeted at providing statistical guarantees and not deterministic guarantees. Approaches similar to Diffserv have been proposed for providing deterministic guarantees [61, 77]. In the SCORE architecture [61], to provide deterministic guarantees without per-flow state management, the per-flow reservation state is carried in the packets of the flows and not stored and maintained in the core. The reservations stated in the packets is then used by the core routers to estimate the aggregate reservation on the links. There are no explicit refresh messages and thus the problems associated with lost or delayed refresh messages do not arise. However, the reservation estimation is dependent on the individual flow behavior and is often inaccurate. It is possible that this kind of estimation based on user-level flows can cause instability and, therefore, it must be decoupled from individual flow behavior. In addition, this approach does not particularly reduce the processing or bandwidth overheads. In [77], a central system, called the bandwidth broker, makes reservations for all the flows in the network, and hence, there is no need for soft-state refresh mechanism and again problems related to lost or delayed refresh messages do not arise. The obvious drawback is that it does not scale to large networks because of its centralized architecture.

Our reservation aggregation schemes differ from those proposed to date. In the aggregation techniques proposed in [8, 23], computing delay bounds in a dynamic environment are not discussed. In BGRP protocol [51], the flows are setup and aggregated along a sink tree for each domain network. The reservation state aggregation in BGRP has some similarities

with our approach, but BGRP is targeted at providing differential services in an inter-domain environment. Our approach provides deterministic guarantees in an intra-domain network or a VPN, and through multipaths provides richer connectivity than a sink tree. The aggregation technique proposed for RSVP [20] are meant for aggregating reservation state of flows within a single multicast group. Our schemes aggregate state of flows belonging to different multicast group and as such is orthogonal to aggregation within RSVP.

Chapter 5

Summary and Future Work

5.1 Contributions

The goal of this thesis has been to provide robust, scalable, efficient and flexible solutions to two critical problems that face IP networks today, namely, Traffic Engineering and Quality of Service. In pursuing this goal, our guiding principle has been to follow a connection-less design methodology, because this is the methodology that contributed to the tremendous success of the Internet. We found that multipaths hold the key to achieving this goal. We explored the use of multipaths, and made considerable progress in understanding how multipaths can enhance the IP network layer unicast routing. We have shown that multipaths combined with routing parameters offer remarkable tradeoffs in performance and scalability in the context of both traffic engineering and guaranteed services. In particular, we have found that connection-oriented technologies, such as MPLS, are not needed to provide scalable and efficient solutions to these problems and their use can be detrimental to the scalability and robustness of current IP networks.

Our first application of multipaths is targeted at traffic engineering in IP networks. The key idea is to approximate the behavior of the minimum-delay routing algorithm[29] and develop a near-optimal routing framework that can be implemented in practical networks. A central component of this framework is a routing protocol that constructs loop-free multipaths and a heuristic that distributes traffic reflecting minimum-delay routing. An inherent problem with any multipath routing approach is that packets can be delivered out-of-order even when the network topology is stable. Though the out-of-order packet delivery of the near-optimal framework may pose a problem with protocols such as TCP, the framework can be very useful in supporting bandwidth intensive protocols that have *relaxed reliability* constraint like the one proposed in IMP [53]. However, to prevent performance degradation of applications that use TCP, the framework is extended to deliver packets in-order for those applications, at least when the topology and the routing parameters are stable. Through simulations we showed that when there is a good mix of flows that need strict and relaxed reliability, hybrid packet forwarding used in the framework can distribute traffic that more closely compares to those seen in minimum-delay routing. The critical design feature of the traffic engineering solution proposed here is that it follows a connection-less approach without using connection-oriented techniques based on MPLS.

For supporting guaranteed services in the Internet, we proposed the SMART architecture based on multipaths rather than connection-oriented mechanism such as MPLS and virtual circuits. The key idea in the SMART architecture is to aggregate flows along multipaths using flow classes. Though the flow classes can be defined in many ways, the ones that satisfy the closure property when distributed over multiple next-hops is difficult to design. The proposed flow classes based on burst-ratio not only satisfy the closure property when flows are split, but are suitable for aggregating along multipaths. Each routing table entry specifies the routing

parameters and the aggregate reservation for the corresponding class-destination pair, and thus the routing state size scales as a linear function of number of destinations. The AGREE protocol designed to maintain the reservations uses amount refresh messages that scales linearly with number of active destinations. It is the first protocol that integrates the reservation mechanism with a multipath routing protocol such that the consistency of reservations is maintained transparently from upper layers in presence of network failures. In our proposed technique for QoS routing, delay constraints are decoupled from bandwidth constraints. In our approach, paths are precomputed based on delay constraints, while bandwidth constraints are enforced at flow setup time. The reason to do this is that, unlike the delay parameter, bandwidth is a consumable network resource whose availability varies on a short time-scale. Users specify only bandwidth and class and not delay as part of their flow request, and the network provides delay bound to the user depending on the class and destination. As a result, the proposed path selection scheme avoids residual bandwidth advertisements and multi-constraint routing that is otherwise needed as in other integrated services architectures. In the SMART architecture, packets can be delivered out-of-order, but this should not be a concern when delay bounds are also provided simultaneously.

The implementation of multipaths requires enhancements to both data and control planes, but the proposed multipath extensions to IP routing are simple and implementable. We have only extended the per-destination routing table entries from specifying a single next-hop to specifying multiple next-hops and the corresponding routing parameters. On the data plane, distributing traffic along multipaths in accordance with routing parameters requires more CPU processing than the simple forwarding based on single next-hop. But the additional time required has complexity of $O(log(K))$, where K is the number of neighbors of the router, which is low to begin with, and given that processing power of the router must grow with

number of outgoing links, the added complexity must be quite acceptable. To support QoS, a fair scheduler at each outgoing link is required. A token-bucket shaper for each class-destination pair is also required at each link. Given that the number of classes will be small and fixed, all these mechanism scale as linear function of number of active destinations. On the control plane, the routing protocols that construct multipaths and routing parameters have similar complexity as current routing protocols. In the SMART architecture, the signaling of flow requests do not require periodic advertisement of link residual bandwidths as the reservations are made on hop-by-hop basis, choosing the widest outgoing link at each hop in the multipath. Also, the reservations are made in-band. The soft-state refresh mechanism of AGREE scales with number of destinations. For these reasons, we believe the proposed multipath mechanism can be easily implemented in today's high-speed networks. Also we believe as long as bandwidth is a scare resource, the performance benefits of using multipaths to improve performance can be justified despite the slight increase in complexity of the routing procedures. Overall, the SMART architecture demonstrates that performance and scalability can be achieved in guaranteed service model without sacrificing the robustness of the IP connection-less architecture.

5.2 Future Work

The following are some new areas of research enabled by the contributions of this thesis.

1. *Multipaths and Multicast Routing :* One promising research area consists of exploring the use of multipaths to improve the network performance in the context of IP multicast routing. The work involves exploring techniques for forwarding packets along multipaths constructed between two join nodes of a multicast tree. For example, the multicast tables can specify the next branch points in the multicast tree rather than the next-hop links.

Once the next branch point node is determined by the multicast routing, it is used as an "intermediate destination" for an encapsulated packet containing the multicast address , and it is this intermediate destination that the multipath routing will use to determine the next-hop link on the path to the intermediate destination. When the packet reaches the branch-point node, the packet is unwrapped to get the multicast address, and the process is repeated.

2. *Multipaths and Differential Services:* This research topic explores the use of multipaths in the context of the differentiated services architecture. The benefits of multipaths are orthogonal to that of queue management; Therefore, multipaths have the potential to improve network performance irrespective of the type of per-hop behavior employed in the routers. Another interesting topic to explore is that of providing differentiated services within the SMART architecture. One possible approach is to have the edge node make an aggregate reservation for several flows originating at that node and provide differentiated services between flows of the aggregate. The responsibility of offering service differentiation between flows in the aggregation is performed at the source edge router. This can potentially improve the bandwidth utilization.

3. *Routing Parameters and MPLS:* Though the techniques in this thesis were presented in the context of connection-less architecture, many of them can be extended to emerging connection-oriented technology such as MPLS. A promising technique is to combine routing parameters with multipaths constructed using MPLS. A *label-switched multipath* (LSMP) is a generalization of *label-switched path* (LSP). First, LSMPs are constructed between each source-destination pair based on such requirements as delay, policies etc. Routing parameters are then assigned at each hop according to some optimization objective and packets are then forwarded along LSMPs according to the routing parameters.

These techniques can be applied to address both QoS and MPLS.

4. *Flow Classes:* The SMART architecture, as currently proposed, defines flow classes based on token bucket parameters. A possible research topic is to explore other ways of forming flow classes that are more efficient in terms of bandwidth usage, and offer tighter end-to-end delay bounds. At the same time the flow classes must also conform to the closure property. We described the parameters for defining flow classes, but left open the question of how to determine the parameters themselves. This requires empirical studies of actual flows in the Internet, which is outside the scope of this thesis. Based on the results, the actual number of flow classes and their parameters must be determined.

A closing Comment

At any point in time, the technologies available at that time determine the algorithms that provide the best cost-performance tradeoff. Our implicit assumption throughout this thesis has been that bandwidth is a scarce resource. With high-speed fiber-optic networks exceeding terabit capacities, the bandwidth available per user is increasing rapidly. It is conceivable that one day bandwidth may be so abundant that the multipath techniques proposed in thesis may not provide any benefits. But, history has often shown that, whenever a resource is available in surplus, new powerful applications will be developed to consume the surplus, and similarly when demand for the resource is great, technology will find a way to fulfill the demand. We can be quite sure, at least for the near future, that there will be times when demand exceeds supply calling for efficient methods to use of the scarce resources. We hope that techniques proposed here will be useful during such times in providing solutions that offer clear cost-performance benefits. We steadfastly adhered to the connection-less model because it has proved to be most effective over time.

Bibliography

[1] R. Albrightson, J.J. Garcia-Luna-Aceves, and J. Boyle. EIGRP-A Fast Routing Protocol Based on Distance Vectors. *Proc. Networld/Interop 94*, May 1994.

[2] D. Awduche and et al. A Framework for Internet Traffic Engineering. *Internet Draft, URL:draft-ietf-tewg-framework-00.txt*, January 2000.

[3] L. Berger and et al. RSVP Refresh Overhead Reduction Extensions. *Internet Draft, draft-ietf-rsvp-refresh-reduct-05.txt*, June 2000.

[4] D. Bersekas and R. Gallager. Second Derivative Algorithm for Minimum Delay Distributed Routing in Networks. *IEEE Trans. Commun.*, 32:911–919, 1984.

[5] D. Bersekas and R. Gallager. Data networks. 2nd ed. *Prentice-Hall*, pages 455–475, 1992.

[6] D. Bersekas and R. Gallager. Data networks. 2nd ed. *Prentice-Hall*, pages 404–410, 1992.

[7] D. Bersekas and R. Gallager. *Data Networks. 2nd Ed.* Prentice-Hall, 1992.

[8] S. Berson and S. Vincent. Aggregation of internet integrated services state. *IWQOS*, 1998.

[9] D. Bertsekas. Dynamic Behavior of Shortest-Path Algorithms for Communication Networks. *IEEE Trans. Automatic Control*, 27:60–74, 1982.

[10] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: An overview. *RFC 1633*, June 1994.

[11] C.G. Cassandras, M.V. Abidi, and D. Towsley. Distributed Routing with Onn-Line Marginal Delay Estimation. *IEEE Trans. Commun.*, 18:348–359, March 1990.

[12] D. Clark. The Design Philosophy of the DARPA Internet Protocols. *Proc. of ACM SIGCOMM*, August 1998.

[13] R.L. Cruz. A calculus for network delay, Part I: Network Elements in isolation. *IEEE Trans. Inform. Theory*, 37(1):114–121, 1991.

[14] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. *Proc. of ACM SIGCOMM*, pages 1–12, 1989.

[15] D. Black et al. An Achitecture for Differentiated Services. *Internet Draft*, May 1998.

[16] D. Katz et al. Traffic Engineering Extensions to OSPF. *draft-katz-yeung-ospf-traffic-01.txt*, Oct. 1999.

[17] G. Apostolopoulos et al. On reducing the processing cost of on-demand QoS path computation. *Journal of High Speed Networks*, 7:77–98, 1998.

[18] G. Apostolopoulos et al. Quality of Serive Based Routing: A Performance Perspective. *Proc. of ACM SIGCOMM*, 1998.

[19] I. Matta et. al. Transient and Steady-State Performance of Routing Protocols: Distance Vectors vs. Link State. *Journal of Internetworking: Research and Experience*, 6:59–87, 1995.

[20] R. Guerin et al. Aggregating RSVP-based QOS Requests. *Internet Draft*, Nov. 1997.

[21] R. Guerin et al. Qos Routing Mechanisms and OSPF Extensions. *Internet Draft*, Jan. 1998.

[22] R. Guerin et al. Scalable QoS Provision Through Buffer Management. *Proc. of ACM SIGCOMM*, 1998.

[23] Rampal et al. Flow grouping for reducing reservation requirements for guaranteed delay service. *Internet Draft:draft-rampal-flow-delay-service-01.txt*, July 1997.

[24] V. Paxson et al. Web page: ita.ee.lbl.gov/html/traces.html. *Lawrence Berkeley National Laboratory*, July 1997.

[25] Y. Bernet et al. An Framework for Differentiated Services. *Internet Draft*, May 1998.

[26] S. Nelakuditi et.al. Adaptive proportinal routing: A localized qos routing approach. *Proc. IEEE INFOCOM*, 2000.

[27] E.W.Dijkstra and C.S.Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11:1–4, August 1980.

[28] D. Ferrari, A. Benerjea, and H. Zhang. Network support for multimedia - a discussion of tenet approach. *Computer Networks and ISDN*, 26:1267–1280, 1994.

[29] R. G. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. *IEEE Trans. Commun.*, 25:73–84, January 1977.

[30] J.J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Trans. Networking*, 1:130–141, February 1993.

[31] J.J. Garcia-Luna-Aceves and J. Behrens. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications*, October 1995.

[32] J.J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Trans. Networking*, February 1997.

[33] J.J. Garica-Luna-Aceves and M. Spohn. Scalable link-state internet routing. *Proc. International Conference on Network Protocols*, October 1998.

[34] L. Georgiadis, R. Guerin, V. Peris, and K. Sivaranjan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. *IEEE/ACM Trans. Networking*, pages 482–501, August 1996.

[35] C. Hendrick. Routing Information Protocol. *RFC*, 1058, june 1988.

[36] P. Humblet and S. Soloway. Algorithm for Data Communication Networks: Part I and II. *Codex Corporation*, 1986.

[37] P. A. Humblet. Another Adaptive Distributed Shortest Path Algorithm. *IEEE Trans. Commun.*, 39:995–1003, June 91.

[38] J. M. Jaffe and F. H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Trans. Commun.*, 30:1758–1762, July 1982.

[39] L. Klienrock. *Communication Nets: Stochastic Message Flow and Delay.* McGraw-Hill, New York, 1964.

[40] D. Kourkouzelis. *Multipath Routing Using Diffusing Computations, M.S. Thesis.* University of California, Santa Cruz, March 1997.

[41] S. Kweon and K. Shin. Providing Deterministic Delay Guarantees in ATM Networks. *IEEE/ACM Trans. Networking*, 6(6):838–850, December 1998.

[42] W. Lai. Capacity Engineering of IP-based Networks with MPLS. *Internet Draft, URL:draft-wlai-tewg-cap-eng-00.txt*, March 2000.

[43] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. *Proc. International Conference on Network Protocols*, October 1997.

[44] P. M. Merlin and A. Segall. A Failsafe Distributed Routing Protocol. *IEEE Trans. Commun.*, 27:1280–1287, September 1979.

[45] J. Moy. OSPF Version 2. *RFC*, 2328, 1998.

[46] P. Pan and H. Schulzrinne. Staged Refresh Timers for RSVP. *Globecom*, Nov. 1997.

[47] P. Pan and H. Schulzrinne. YESSIR: A Simple Reservation Mechanism for the Internet. *Computer Communications Review*, 29(2), April 1999.

[48] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Trans. Networking*, 1:344–357, June 1993.

[49] R. Perlman. Fault-tolerant broadcast of routing information. *Computer Networks and ISDN*, 7, 1983.

[50] C. Pornavalai, G. Chakraborty, and N. Shiratori. QoS based routing algorithm in Integrated Services Packet Networks. *Journal of High Speed Networks*, 7:99–112, 1998.

[51] P.Pan and et al. BGRP: A Framework for Scalable Resource Reservation. *Internet Draft, draft-pan-bgrp-framework-00.txt*, July 2000.

[52] B. Rajagopalan and M. Faiman. A Responsive Distributed Shortest-Path Routing Algorithm with Autonomous Systems. *Internetworking: Research and Experience*, 2:51–69, March 1991.

[53] S. Raman. ITP: An Image Transport Protocol for the Internet. *Proc. Intl. of Network Protocols*, Nov. 2000.

[54] M.I. Reiman and A. Weiss. Sensitivity analysis for simulations via likelihood rations. *Proc. 1986 Winter Simulation Conf.*, pages 285–289, 1986.

[55] A. Segall. The Modeling of Adaptive Routing in Data Communication Networks. *IEEE Trans. Commun.*, 25:85–95, January 1977.

[56] A. Segall. Optimal distributed routing for virtual line-switched data networks. *IEEE Trans. Commun.*, 27:201–209, January 1979.

[57] A. Segall and M. Sidi. A Failsafe Distributed Protocol for Minimum Delay Routing. *IEEE Trans. Commun.*, 29:689–695, May 1981.

[58] A. Shaikh, J. Rexford, and K. Shin. Efficient computation of Quality-of-Service routes. *NOSSDAV*, 1998.

[59] J. Spinelli and R. Gallager. Event Driven Topology Broadcast without Sequence Numbers. *IEEE Trans. Commun.*, 37:468–474, 1989.

[60] D. Stiliadis and A. Varma. Rate-Proportional Services: A Design Methodology for Fair Queuing Algorithms. *IEEE/ACM Trans. Networking*, April 1998.

[61] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. *Proc. of ACM SIGCOMM*, Sept. 1999.

[62] K. Takashima and et al. Concept of IP Traffic Engineering. *Internet Draft, URL:draft-takashima-te-concept-00.txt*, October 1999.

[63] A. Tanenbaum. Computer networks. 3rd ed. *Prentice-Hall*, pages 357–358, 1996.

[64] D. Thaler. Multipath issues in unicast and multicast next-hop selection. *Internet Draft, URL:draft-thaler-multipath-05.txt*, Feb 2000.

[65] C. Villamizar. MPLS Optimized Multipath. *Internet Draft, URL:draft-ietf-mpls-omp-00.txt*, March 1998.

[66] C. Villamizar. OSPF Optimized Multipath. *Internet Draft, URL:draft-ietf-ospf-omp-00.txt*, March 1998.

[67] S. Vutukury and J.J. Garcia-Luna-Aceves. A Scalable Architecture for Providing Deterministic Guarantees. *Proc. of ICCCN*, Oct. 1999.

[68] S. Vutukury and J.J. Garcia-Luna-Aceves. A Simple Approximation to Minimum Delay Routing. *Proc. of ACM SIGCOMM*, Sept. 1999.

[69] S. Vutukury and J.J. Garcia-Luna-Aceves. A Multipath Framework Architecture for Integrated Services. *Proc. IEEE GLOBECOM*, 2000.

[70] S. Vutukury and J.J. Garcia-Luna-Aceves. A Traffic Engineering Approach to Minimum Delay Routing. *Proc. of ICCCN*, Oct. 2000.

[71] L. Wang and et al. RSVP Refresh Overhead Reduction by State Compression. *Internet Draft, draft-wang-rsvp-state-compression-03.txt*, March 2000.

[72] Y. Wang and Z. Wang. Explicit Routing Algorithms for Internet Traffic Engineerin. *Proc. of ICCCN*, pages 582–588, 1999.

[73] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14:1128–1234, 1996.

[74] W. T. Zaumen and J.J. Garcia-Luna-Aceves. Loop-Free Multipath Routing Using Generalized Diffusing Computations. *Proc. IEEE INFOCOM*, March 1998.

[75] H. Zhang and D. Ferrari. Rate-Controlled Service Disciplines. *Journal of High Speed Networks*, Feb. 1994.

[76] L. Zhang and et al. RSVP: A New Resource Reservation Protocol. *IEEE Communications Magazine*, 31(9):8–18, 1993.

[77] Z. Zhang and et al. Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services. *Proc. of ACM SIGCOMM*, pages 71–83, 2000.